NBSIR 83-2713

# Time of Day Control and Duty Cycling Algorithms for Building Management and Control Systems

U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
National Engineering Laboratory
Center for Building Technology
Building Equipment Division
Washington, DC 20234

June 1983

Sponsored by:
Office of Buildings and Community Systems
U.S. Department of Energy
U.S. Navy Civil Engineering Laboratory
U.S. Department of Defense

NBSIR 83-2713

# TIME OF DAY CONTROL AND DUTY CYCLING ALGORITHMS FOR BUILDING MANAGEMENT AND CONTROL SYSTEMS

William B. May, Jr.

U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
National Engineering Laboratory
Center for Building Technology
Building Equipment Division
Washington, DC 20234

June 1983

Sponsored by:
Office of Buildings and Community Systems
U.S. Department of Energy
U.S. Navy Civil Engineering Laboratory
U.S. Department of Defense

**U.S. DEPARTMENT OF COMMERCE, Malcolm Baldrige, *Secretary***
**NATIONAL BUREAU OF STANDARDS, Ernest Ambler, *Director***

## ABSTRACT

Software is an important component of building management and control systems (BMCS). This report describes concepts, algorithms, and software used in BMCS components developed in the NBS building systems and controls laboratory. The concepts and basic algorithms for time of day (scheduled start/stop) control and duty cycling of electrical equipment in building heating, ventilating, and air conditioning systems are presented. Time of day control results in control events occurring at predetermined times of the day on selected days of the week. Duty cycling is the periodic turning off and on of loads ,usually electrical, to reduce energy consumption under part heating and cooling load conditions. Considerations for use of duty cycling with other control strategies such as demand limiting, selection of duty cycling parameters, and dynamic adjustment of duty cycling, are discussed. All algorithms presented were implemented in software on a specific BMCS, and the actual computer programs used are presented as examples.



KEY WORDS: Building Management and Control Systems (EMCS,BMCS); Computer Control; Control Algorithms; Control Software; Duty Cycling; Energy Management; Heating, Ventilating and Air Conditioning (HVAC); Scheduled Start/Stop; Time of Day Control.

TABLE OF CONTENTS

## LIST OF FIGURES



## LIST OF TABLES

# 1. INTRODUCTION

A computer-based building management and control system (BMCS) relies heavily on computer software to fully utilize the heating, ventilation, and air conditioning (HVAC) equipment available in the building. Much HVAC control software is available in proprietary or system dependent form, usually supplied with a BMCS system purchased from a particular manufacturer. However, if software source listings are not supplied, the HVAC designer or BMCS owner will not know if the BMCS system is actually operating according to the specifications of the designer or owner. Without knowledge of what control algorithms the system is using, it is difficult to compare the operation of one BMCS system to another. Even if source listings are made available, there is little HVAC control algorithm software in the public domain to use for baseline comparison purposes. This report describes concepts and non-proprietary algorithms used in control software that was developed at the National Bureau of Standards (NBS) for use in the NBS building management and controls laboratory.

Building management and control system algorithms fall into at least two categories. Direct control refers to strategies that control the building equipment directly, implementing closed loop control of valves and dampers, and replacing conventional pneumatic or electronic local analog control methods. Direct control is most often used to maintain a building zone or fluid stream at a setpoint temperature or humidity. Supervisory or management control strategies control the building in a broader sense, managing equipment as a function of variables such as electrical demand, time, weather conditions, occupancy, and the overall state of the building in order to maximize occupant comfort and minimize energy consumption and operating cost.

Modern building management and control systems are usually distributed among a heirarchy of several physically separated control computers. Control algorithms are usually aimed at operation at some particular level of the system. For example, the tri-services specification defines four levels in a tree-tree shaped hierarchy for a large BMCS [1]. In order from the highest level, these are the central level (CCU), the communication controller level (CCC), the distributed control level (FID), and the data gathering and conditioning level (MUX). The distributed control level is occupied by equipment given the name FID for field interface device or RPU for remote processing unit. It is at this level that the software described in this report is intended to operate.

This report is concerned with two types of supervisory control, commonly called time-of-day control and duty cycling. These types of control are on the low end of the scale as far as complexity of control are concerned. However, their implementation is not necessarily simple. The basic concepts and algorithms for these two types of control are presented in sections two and three of the report. In section four, concrete examples of the two types of control algorithms are given in the form of the computer software used in an actual BMCS developed at NBS.

## 2. TIME-OF-DAY CONTROL

Time-of-day control is the starting or stopping of equipment or initiation of events at predetermined times of the day on selected days of the week. The absolute time that the event occurs (for example, 5 PM on Monday) must be close to the predetermined time for the control strategy to be successful. Timed control of events can also be relative, causing an event to occur after a delay from the present time. Relative time control will not be discussed here.

### 2.1 Basic Concepts

Events initiated by time of day control can be simple starting or stopping of equipment such as pumps, fans, or compressors. This may be referred to as scheduled start/stop. The loads to be started or stopped can be selected along with the time of day and day of the week that the loads are to be turned off or on.

Events to be controlled may also be more complex. An example of a complex event might be a startup sequence which turns on ten fans with delays of five seconds between the starting times of the fans. A time-of-day controlled event might also consist of reading several meters, performing calculations and sending the results to a printer.

Time-of-day control is usually applied to shutting down a building during unoccupied periods in order to save the energy that would have been used to keep fans and lights operating throughout the unoccupied period. The same type of control can be used to turn on building equipment before the building is to become occupied.

### 2.2 Basic Algorithm

The basic algorithm for time-of-day control is very simple, as shown by the flow chart in figure 1. If an event is to be under time-of-day control, the flow chart is entered and the desired time for the event is determined. Then the current time must be determined. If the current time and the event time are the same, then the event is caused to occur. Otherwise, no action is taken and the algorithm is exited. This algorithm assumes that a means (ie. software) is available to read a real-time clock/calendar device, that the time for an event to occur can be obtained from a storage area in computer memory or from an input device, and that a means exists to cause the event to occur.

The algorithm in figure 1, to be useful, must be executed periodically throughout the day until the time for the event is the same as the current time. The time between executions of the algorithm must be equal to the allowable error between the event time and the real time. The allowable error

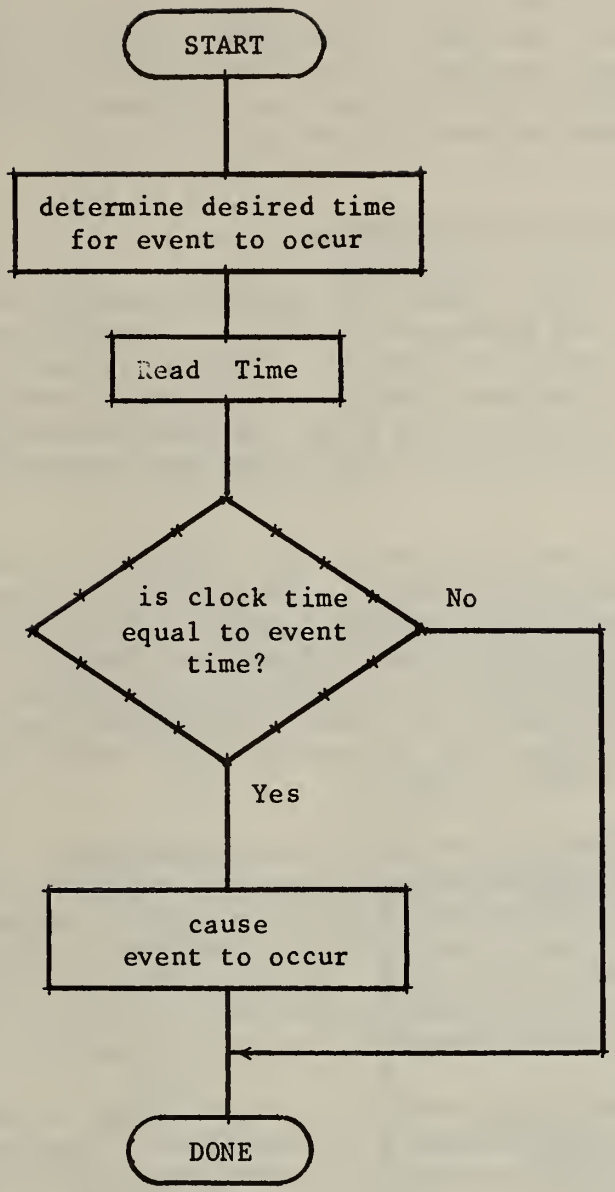


figure 1. Basic algorithm for time-of-day control

is usually one minute, so that an event scheduled for 9:45 might happen anytime between 9:45 and 9:46. The algorithm, in this case, would be executed once every minute. When the comparison between times is made, the term "equal" is used loosely to mean that the difference between the two times is less than the maximum allowable error.

## 2.3 Time-of-day Control Considerations

The algorithm of figure 1 does not take into account many complications that might occur in a real BMCS system. The most obvious of these is the need to schedule several events to occur at different times on different days. To improve the utility of the algorithm, the simple basic time-of-day algorithm would have to be replaced by a more complex algorithm consisting of an execution of the basic time-of-day algorithm for each event that was to be scheduled.

Another consideration for time-of-day control is that several concurrent strategies might be attempting to control the same loads or equipment. The event that the time-of-day control algorithm is scheduled to execute might call for the activation of a fan that has been manually shut down for service. A solution is to use a priority system for controlling equipment. This means that each controller (including manual control) that can affect a load is given a priority rating, and no controller of lesser priority can override a controller with a higher priority.

If there is a problem with the BMCS remote equipment, it is possible that when the time-of-day controller issues a command request to turn on or off a load, the load might never be affected because a remote actuator is malfunctioning. For this reason it may be necessary to include feedback capabilities in a time-of-day control algorithm. This would mean waiting for a report or indication that a load had been successfully controlled and if no report came, causing an alarm to be displayed to the system operator.

In a large BMCS system, the amount of time required to execute the time-of-day control software might become a consideration. If the software is executed once per minute in a microprocessor-based controller, and the check of times for a very large number of sheduled events approaches one minute, this would not leave time for other system functions. It is possible to avoid this problem by using a scheme which only executes the time-of-day control software as often as necessary to control the scheduled events.

## 2.4 More Complex Time-of-day Algorithm

A more complex algorithm than the one in figure 1 is given in figure 2. Three of the considerations mentioned in section 2.3 have been added to the algorithm. The clock-reading function has been moved to the beginning of the algorithm, as it is assumed that the algorithm can be executed rapidly enough

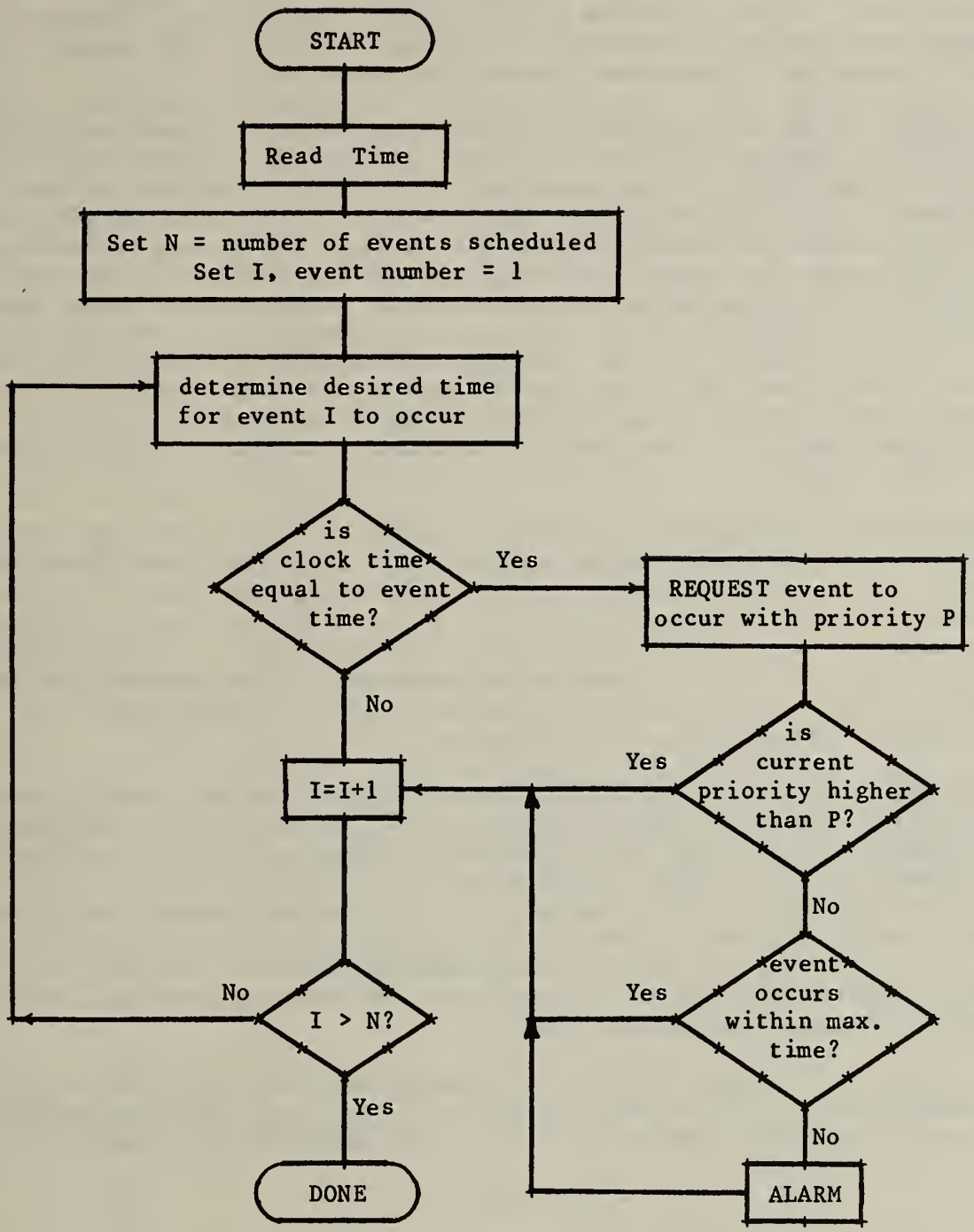


figure 2. More complex algorithm for time-of-day control

so that the difference between the real clock time and the clock time measured at the start of the algorithm is negligible. The ability to schedule multiple events has been added in the form of a cycle, once for each event, through the basic comparison of the event time to the clock time.

In figure 2, The simple block labelled "cause event to occur" in the algorithm of figure 1 has been replaced by a more complex chain of decisions, beginning with a "request" for the event to occur. The occurence of an event is assummed to be incomplete until some evidence of completion is available. The control request is made with a priority level. The control software that actually causes the event to occur should check the priority that the time-of-day controller is using to request that the event occur. If the time-of-day controller's priority is too low, the request will be rejected. If the priority is high enough, an attempt will be made to initiate the event. If some feedback that the event has occurred is not made available to the time-of-day controller within a certain maximum time interval, an alarm will be generated. Otherwise, the algorithm continues checking for other events whose times have come.

There is an alternative to verifying that an event has occurred within a maximum time interval before moving on to the next event. The alternate method is to delay the verification of event occurrence until after all events times have been compared to the clock time. A logical loop, similar to the one used to compare clock time to event times, can be used to verify that all events have occurred. This method is preferable if the maximum time delay for verification of events is large compared to the execution time of the time-of-day control algorithm.

Some means must be used to store the information on what time the event is to be initiated, what software is to be used to initiate this event, and other information such as the identification of loads to turn on or off in the case of scheduled start-stop. One method to accomplish this is the use of a software table. In such a table, stored in computer memory, each row of the table would represent an event. The columns of the table would contain the times of day, days of the week, and other information associated with events. The arrangement and method of retrieving information from the table will be specific to a particular application.

An example of a complex time-of-day control algorithm is given in section 4.2, illustrating the use of one arrangement of an event information table. The example also uses a scheme for reduction of total algorithm execution time.

## 3. DUTY CYCLE CONTROL

Duty cycling is used to reduce energy consumption by equipment which may be periodically shut down for short periods of time without degradation of the building environment or critical building functions. This type of equipment is usually associated with "operating energy", defined as energy consumed to transport heat transfer fluids (such as air and water) around a building but not to heat or cool these fluids. The equipment will usually be electrical in nature with starting current requirements larger than operating current values. Fans and pumps are typical duty cycled loads. If a duty cycled load normally runs at a constant power input (e.g. single-speed fan) then the amount of energy saved by duty cycling is approximately equal to the fraction of total time that the duty cycled load is off, multiplied by the energy consumption of the load if it is not cycled.

The use of duty cycling control is justified because HVAC systems are usually not operated at peak design conditions. At off-peak conditions, the full flow provided by a fan or pump may not be required to maintain space conditions. To be used for duty cycling, a piece of equipment should have sufficent capacity to return conditions to normal levels after any drift which occurs when the equipment is off.

### 3.1 Basic Concepts

There are five parameters which must usually be defined for any load to be duty cycled. Three of these five parameters are illustrated in figure 3. When a load is duty cycled, as for example load 1 in figure 3, the load, which is on at the start of the interval shown in the figure, is turned off at a certain point. After a delay, which is referred to as the off-period, the

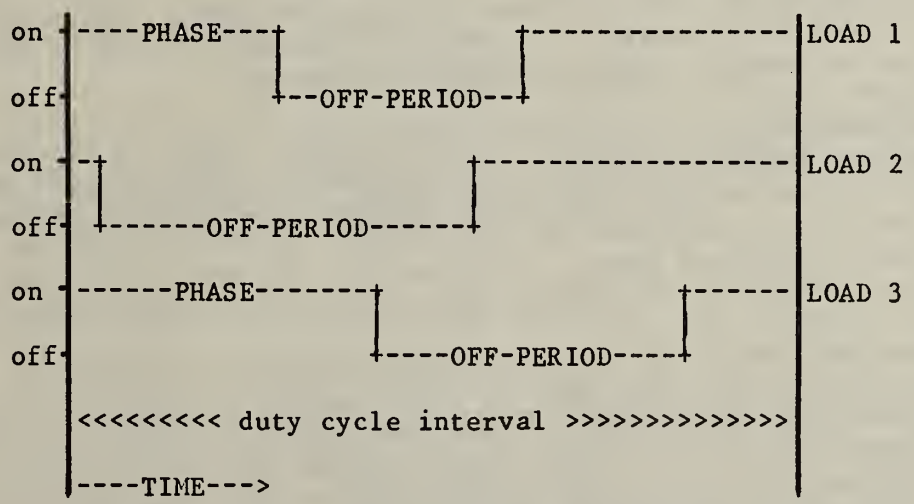


figure 3. parameters for duty cycling

load is turned back on again. This "on-off-on" cycle is repeated continously for duty cycling. The length of a single "on-off-on" cycle is defined as the duty cycle interval. The delay between the beginning of the duty cycle interval and the turning-off of a load will be referred to as the phase for that load.

If several loads are being duty cycled at the same time, such as loads 1, 2 and 3 in figure 3, they may or may not have the same off-period. When more than one load is duty cycled, the most important consideration is to avoid turning two loads back on at the same time. If two loads are activated simultaneously, and the loads are electrical, then the high starting currents will result in a large electrical demand peak which could be avoided if the loads are not started at the same time. Therefore, it is desirable to separate the turn-on times of equipment. Another consideration is that certain loads may be interrelated, and it may be undesirable to have both loads off (or on) at the same time. If the off-periods are specified for a number of loads to be duty cycled, then the phase for each load can be individually adjusted to prevent simultaneous activation of loads.

In addition to the three parameters in figure 3, a minimum on-period and a minimum off-period may be specified. These times are primarily specified to avoid damage to equipment due to excessive cycling. The minimum on-period and off-periods must be in accordance with either manufacturers recommendations or any applicable standards.

### 3.2 Basic Duty Cycling algorithm

The basic algorithm for duty cycling a single load is displayed in figure 4. The algorithm must be executed periodically at a frequency equal to the inverse of the duty cycle interval. The phase and off-period values for the load are obtained as the first step of the algorithm. There is then a delay for a time interval equal to the phase. The method used to cause the delay depends on the control system computer architecure.

After the phase delay, the load is turned off. A load to be duty cycled will in most cases be controlled through a digital output, which can be used to remove and restore electric power to the load. Application dependent software is required to actually turn off the digital output. After the deactivation of the load is another delay, equal to the off-period for the load. After this delay the load is then turned on, and the single duty cycle is complete.

Some means must be used to store the information on what the off-period, phase, and other parameters are for each load to be duty cycled. One method to accomplish this is the use of a software table. In such a table, stored in computer memory, each row of the table contains information for one load. The columns of the table contain the duty cycling parameters for the load. The structure of the table and the method used to store and retrieve information in the table will be specific to a particular application.

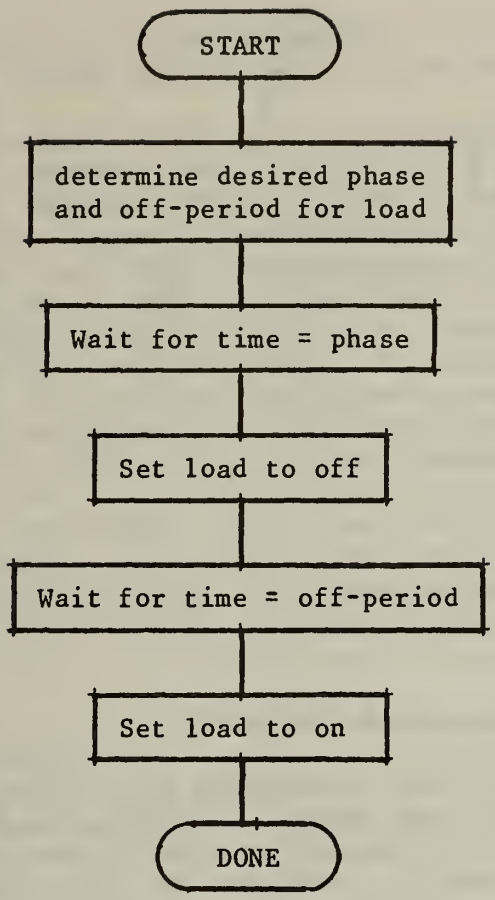


figure 4. Basic algorithm for duty cycling control

### 3.3 More Complex Duty Cycling Algorithm

The algorithm in figure 4 is useful for illustrating the basic concept of duty cycling but would not be practical for use in a real BMCS. An improvement to the algorithm is to allow a number of loads to be concurrently duty cycled. This can be done by creating a software loop which executes an algorithm, such as the one in figure 4, for each load to be duty cycled.

To extend the versatility of the duty cycle algorithm, the last four steps in figure 4 can be replaced by a single step, which will be to make a request to a software routine which will be called the load controller. The load controller can be used to control the timing of the off-cycle for the load. The load controller can be given the values of phase and off-period, along with the load identification number. It will manage the book-keeping required to turn the load off and on at the correct times. The exact structure and actions of the load controller are dependent on the architecture of the BMCS computer, but if properly designed, the load controller can be called several

times in rapid succession for different loads and will still turn all of the loads off and on at the proper times in the future.

It has been assumed so far that duty cycle intervals, phases, and off-times are expressed in units of absolute time. However, use of absolute times in the algorithm can make it inflexible, for example, if the duty cycle interval must be changed. An alternative is to express the duty cycle interval in absolute time units and then express the phase and offset in terms of a percentage of the duty cycle interval. This allows the duty cycle interval to be changed without having to change the specifications for the phase and off-period. The only penalty for this generalization of time is that the phase and off-period must be converted to absolute time units when the duty-cycle algorithm is executed by multiplying the percentage of the duty cycle interval by the duty cycle interval time.

Figure 5 shows the revised duty-cycle algorithm incorporating the changes discussed above. This algorithm is more useful than the algorithm of figure 4 but does not take into account complications existing in real BMCS systems. Such complications are discussed in the following sections.

### 3.4 Duty Cycling Used With Other Strategies

In most cases, a duty cycling algorithm will be required to control loads in cooperation with other load control algorithms, such as scheduled start/stop, optimum start/stop, and demand limiting. This requires that there be a priority scheme for control of outputs, so that a load which has been turned off for the night is not turned on by the duty cycler, or a load turned off by the demand limiting software is not turned on again by the duty cycler before the demand peak has been lowered.

The assignment of priority to control algorithms is somewhat application dependent, but some generalized observations may be made. If a scheduled start/stop algorithm is active at the same time as duty cycling control, the scheduler algorithm should have the higher priority if a load is currently off. During unoccupied periods, a scheduler will typically stop equipment to conserve energy. It would not be desirable for the duty cycler to be able to control equipment that had been stopped for such a reason. On the other hand, when the scheduler has turned on equipment for the occupied period, then the equipment could be duty cycled. This implies that the scheduler should have a lower priority than the duty cycler when the load is currently on.

Demand limiting is another supervisory type of control that cuts off loads to reduce electrical demand peaks. If a demand limiting algorithm were active at the same time as the duty cycling algorithm, the demand limiting algorithm could be attempting to control the same loads as the duty cycler is controlling. Since the purpose of the demand limiter is to prevent electrical demand peaks, and the purpose of the duty cycler is only to lessen operating time of equipment, it is usually desirable to let the demand limiting

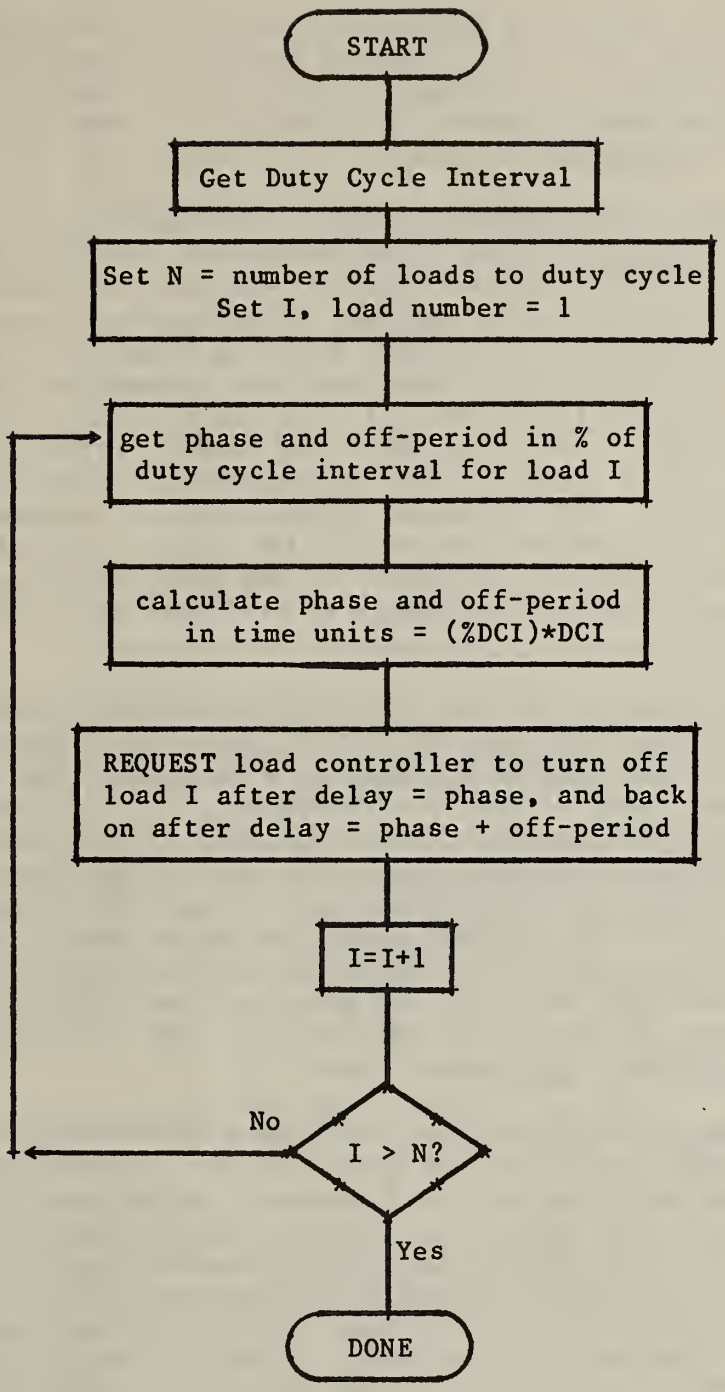


figure 5. improved algorithm for duty cycle control

algorithm have priority in controlling a load currently being duty cycled. Priorities can be used to arbitrate conflicts between the two control algorithms so that the demand limiter always gets preference and can accomplish its purpose. However, this raises at least three problems in the design of the duty cycling algorithm.

The first problem with combination duty cycling/demand limiting that must be overcome is assuring that the duty cycler will not turn off a load for the full off-period if the demand limiting algorithm has already turned off the load for some time period within the duty cycling interval. A solution to this is to keep a record of the time interval since the load was last off and the interval since the load was last on. When the load controller is requested to duty cycle a load, it must first verify that the duty cycle priority is greater than the priority under which the load was last controlled. If the priority is low enough, and it is determined that the load has been off for a certain time interval since the last duty cycle, the off-period of the load can be reduced by the same interval. The off-period reduction is accomplished by delaying the turning-off of the load, while keeping the time at which the load is to be turned on the same.

A second consideration is that the demand limiting program may have just turned back on a load that the duty cycler is about to turn off. Because most equipment has a finite start-up time it may be undesirable to turn off a piece of equipment just after it has been turned on [2]. A minimum on-period may be specified to avoid damage. The record-keeping scheme mentioned previously must be implemented in order to know how long it has been since the load was turned on. When the load controller is requested to cycle the load, if the time interval since the load was turned on is too small, either the turn-off time for the load may be delayed (reducing the off-period) or the phase may be increased by shifting turn-off and turn-on times (off-period not changed). The disadvantage of the latter approach is that the new turn-on time may coincide with another load turn-on time.

Finally, a piece of equipment may also have a minimum off-period. A minimum off-period check should not be needed often, because neither the demand limiter or the duty cycler would turn on a load that the other control algorithm had just turned off. The duty cycler could not turn on a load turned off by the demand limit algorithm because of priority, and the demand limiter would not normally override the duty cycler to turn on a load (the demand limiter would normally override the duty cycler and turn off a load). The minimum off-period check is needed if the duty cycle parameters are being adjusted by another algorithm to prevent the off-period from becoming too small.

Figure 6 presents a simple load controller algorithm, which can be considered an expansion of the load controller block in figure 5. Values of phase, off-period, priority, and load identification are passed to the load controller from the duty cycler. The algorithm in figure 6 implements the priority,

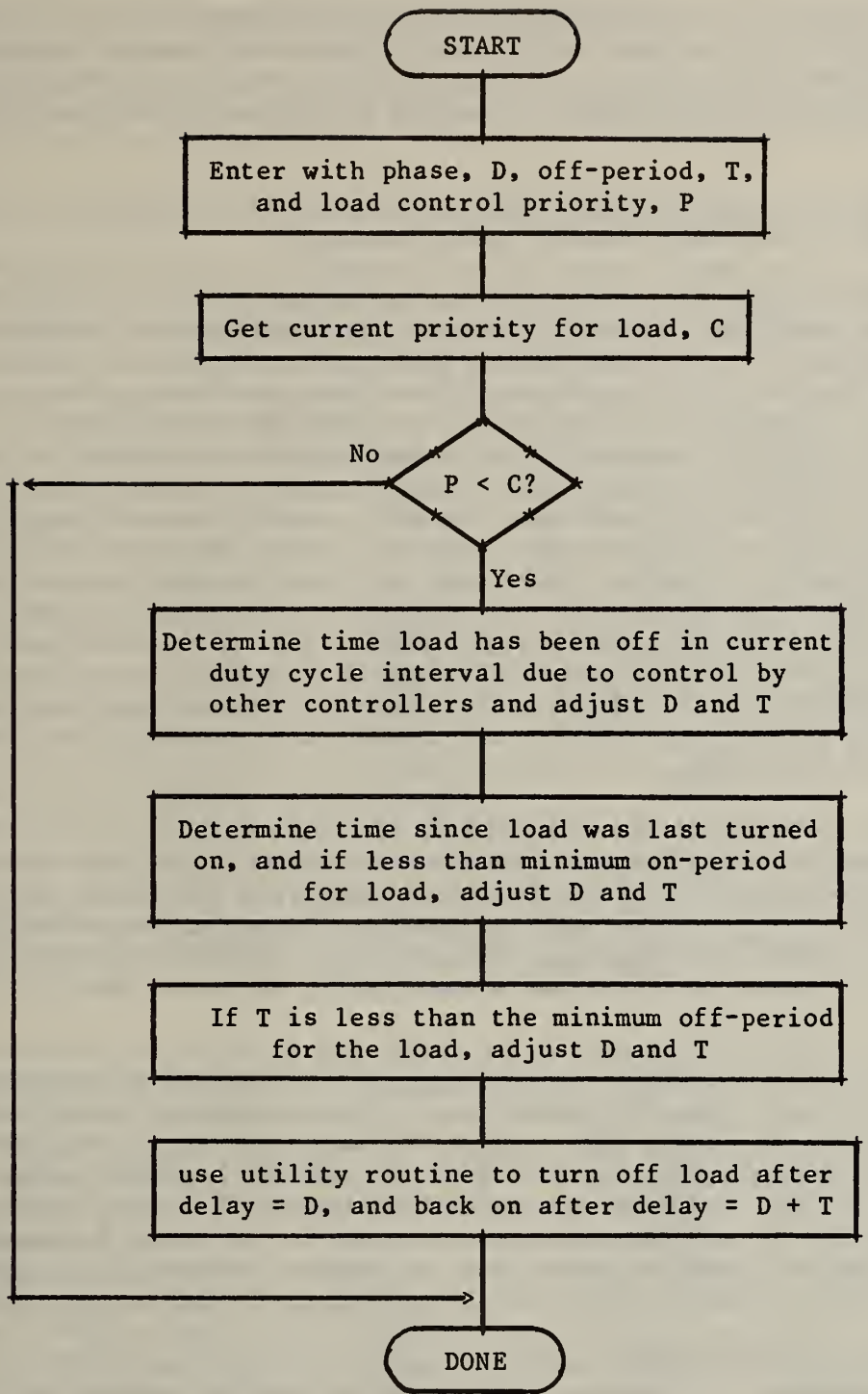


figure 6. basic load controller algorithm for duty cycle control

minimum off-period, and minimum on-period checks previously described. The exact implementation of the last block in the algorithm is dependent on the control system architecture and multi-tasking utility software available. Ideally, whatever is used should result in a minimum of overhead for the load controller.

### 3.5 Selection of the Duty Cycling Phase Parameter

Proper selection of the duty cycling parameters, phase, off-period, and interval, is important to maximize energy conservation and comfort and avoid equipment damage. The off-period and duty cycle interval for a load are very system and application dependent. These two parameters are discussed in section 3.6. Although it is important to consider the complete system when selecting duty cycle parameters, it is possible to select the off-period and interval for individual loads without regard to other loads being duty cycled. The selection of duty cycle phases, however, must be made by considering all duty cycled loads simultaneously, since the phase is the primary parameter that is adjusted to determine the moment in time that the load is to be turned on after being off. Indiscriminate activations of several loads at the same time will cause high electrical demand peaks. When a group of loads are to be duty cycled, their off-periods will usually be a function of the state and type of the HVAC system. The phases, however, can be assigned almost any value as long as the sum of the phase and the off-period is not longer than the duty cycle interval.

If duty cycle parameters are selected at the central control unit (CCU) level, the selection can be made manually by the operator or by computer software. It is not absolutely necessary to use an algorithm to set up the duty cycle parameters if the operator is willing and has the ability to carefully arrange load turn-on times, taking into account any relationships between loads. Otherwise, an algorithm which can automatically set up a table of duty cycle parameters is necessary.

If the duty cycle parameters are modified, either by command from the CCU or by a FID routine during FID operation, it is necessary to adjust the duty cycle parameters to insure that a load turn-on time has not shifted enough to coincide with another load turn-on time. At the FID level, no manual operator adjustment of duty cycle parameters may be possible, and an algorithm is definitely necessary if the parameters are to be adjusted dynamically. The FID level algorithm may not have to be as sophisticated as an algorithm used at the CCU level, since the FID will be adjusting the table, not creating it.

An algorithm used for determining phases for duty cycling loads can have any level of complexity. A simple version might assign phases for a group of loads so that the turn-on times for the loads are evenly spaced across the duty cycle interval. More complex versions would take into account relationships between loads. One relationship could be, for example, that two individual loads should not be off at the same time. The algorithm could

assign phases to the loads to avoid causing the loads to be simultaneously off. Another example of a relationship could be two loads that must be on at the same time or off at the same time. An example of a fairly general algorithm for assigning duty cycle phases is given in section 4.4.

### 3.6 Selection and Dynamic Adjustment of Duty Cycle Off-Period and Interval

Duty cycled devices can be equipment such as exhaust fans and utility pumps, and for these applications, selection of duty cycle off-periods and intervals is usually valid for all states of load of the mechanical system. Duty cycling may also be used on fans and pumps associated with air handling equipment supplying conditioned air to the building space. When air handler equipment is duty cycled, the characteristics of the air handler must be taken into account. It can be assumed that turning on and off an air handler fan will cause oscillation in the space condition. The amplitude and phase of the oscillation will be dependent on the air handler load, the system characteristics and the amount of cycling. This oscillation of space condition should be assummed to cause changes in the amount of ventilation air as well as the wet and dry bulb temperatures. Problems with the closed loop control of the heating and cooling coils might arise as the air flow through the air handler fluctuated from zero to full flow. Due to these considerations, it is desirable to match the amount of duty cycling to the state of the air handling system and dynamically adjust the duty cycling parameters as a function of air handler load, the quantity of outside air entering the space, or some other parameter.

The problem of selecting off-periods and duty cycle intervals may be broken into two parts: one being the determination of the best amount of duty cycling at a specific design point, and the other being what variable to relate duty cycling to and how to adjust the duty cycle parameters as a function of this variable.

There are a number of possible approaches to the determination of optimum duty cycling for an air handler at a design point. A complete answer requires experimental investigation or simulation of the air handler/space system which is at present beyond the scope of this report. One simplified approach is to consider a duty cycled air handler as being a crude implementation of variable air volume (VAV). This is reasonable since duty cycling effectively reduces the average amount of air entering the conditioned zones. A duty cycled system, unlike a VAV system, would not provide zone-dependent volume control. It would reduce the suppy air volume to all zones served by an air handler. This approach implies that duty cycling is not a useful technique for use on a true VAV air handler system and should only be applied to constant volume systems. The question of whether this is true must be answered by more detailed investigation.

If a duty cycled air handler can be considered to be a simplified VAV air handler, then the required supply air volume is a function of space load, and

the off period of the air handler can be varied to give an average air flow equal to the desired supply air volume. This average air flow volume should be larger than the supply volume that an equivalent VAV system would supply because of the possible oscillations in space temperature caused by the duty cycling. Such oscillations reduce space comfort if allowed to become large.

A common method of adjusting HVAC system setpoints that should be determined as a function of space conditioning load is to assume that the space load and outside air temperature are proportional, and adjust the setpoint to follow changes in outside air temperature. If the requirements for latent heat removal, internally generated heat removal, and ventilation of the space are steady enough to be considered constant, then the space load could be considered to be approximately a function of outside air temperature. This assumes that transmission and infiltration losses and gains are a linear function of the inside-outside temperature difference. Using this approximation, the off-period of duty cycling could be varied as a function of outside air temperature for a given supply air temperature setpoint. Alternatively, a method of determining the actual space conditioning requirements could be used to vary the duty cycle off time.

If the design point for duty cycling is chosen as the condition of minimum space load where the supply air requirement is lowest, then at this condition the duty cycling off-period would be a maximum. At off-design conditions, the off-period could be reduced as a function of outside air temperature (or some other parameter) to reach a minimum at large space loads. Large space loads could be indicated by very high or very low outside air temperatures. Figure 7 illustrates the concept. The off-period is modulated downward on either side of the design point to zero at high and low limits. To create a one-sided ramp rather than a two-sided one, either the low or high limit may be made equal to the analog value at the design point. The advantage of choosing the design point as the point of maximum off-period is that no algorithm to

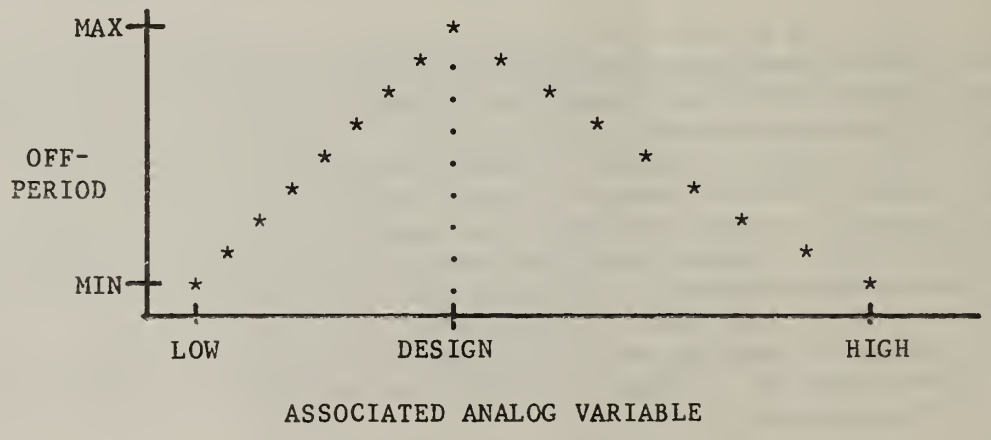


figure 7. Adjustment of duty cycling off-period as a function of an analog parameter

juggle duty cycle phases is required, since the off-period is only reduced in size from the design condition. The phase may be increased and the on-time remains the same.

If the duty cycle design point is chosen to be the condition of minimum space load, and outside air temperature is assumed to be an indicator of load, there is an outside air temperature where the load is a minimum. The duty cycle design off-period can then be chosen for this condition. Care must be taken in determining the design duty cycle off-period because at low supply air volumes, the space requirements for latent heat removal and minimum ventilation (outdoor air) may be dominant.

If duty cycling is used on a system which has a minimum outside air damper which is normally fixed at a position based on a percentage of a constant volume of supply air, the minimum outside air damper controls may have to be modified. If they are not, then at low space loads when the duty cycle off-period for the fan is a maximum, and the outside air is also at a minimum, insufficient ventilation air may be reaching the space.

A design duty cycle interval must also be chosen. In practice, to simplify the control algorithm, the duty cycle interval may be fixed at the design value, and not varied with load. The duty cycle interval should be chosen to minimize oscillation of space conditions. A duty cycle interval that is too long will result in relatively long off-periods and on-periods. The long off-periods may produce excessive drifting of space conditions. On the other hand, a short duty cycling interval will cause more cycling of the equipment and lack of ability to adequately modulate the off-period. For example, if a duty cycle interval is five minutes, if the maximum off time for a load is two minutes, and if the minimum on time is two minutes, then the load off-period could only be varied between two minutes and three minutes (three minutes off plus two minutes on totals five minutes). However, if the duty cycle interval is fifteen minutes, the off-period could be varied between two and thirteen minutes.

Another approach to dynamic adjustment of duty cycling parameters would be the use of a closed loop controller. If the amount of oscillation in space conditions could be quantified, the duty cycle off-period could be adjusted using a proportional-integral control algorithm, or equivalent, to select the maximum off-period that would result in acceptable oscillation of space conditions.

Figure 8 shows the duty cycle algorithm of figure 5 with an additional block for adjusting the duty cycle algorithm. The duty cycle parameters that are retrieved for the load are the design values. These are adjusted using an approach such as the two mentioned here. The load controller is then given the modulated phase and off-time for the load.

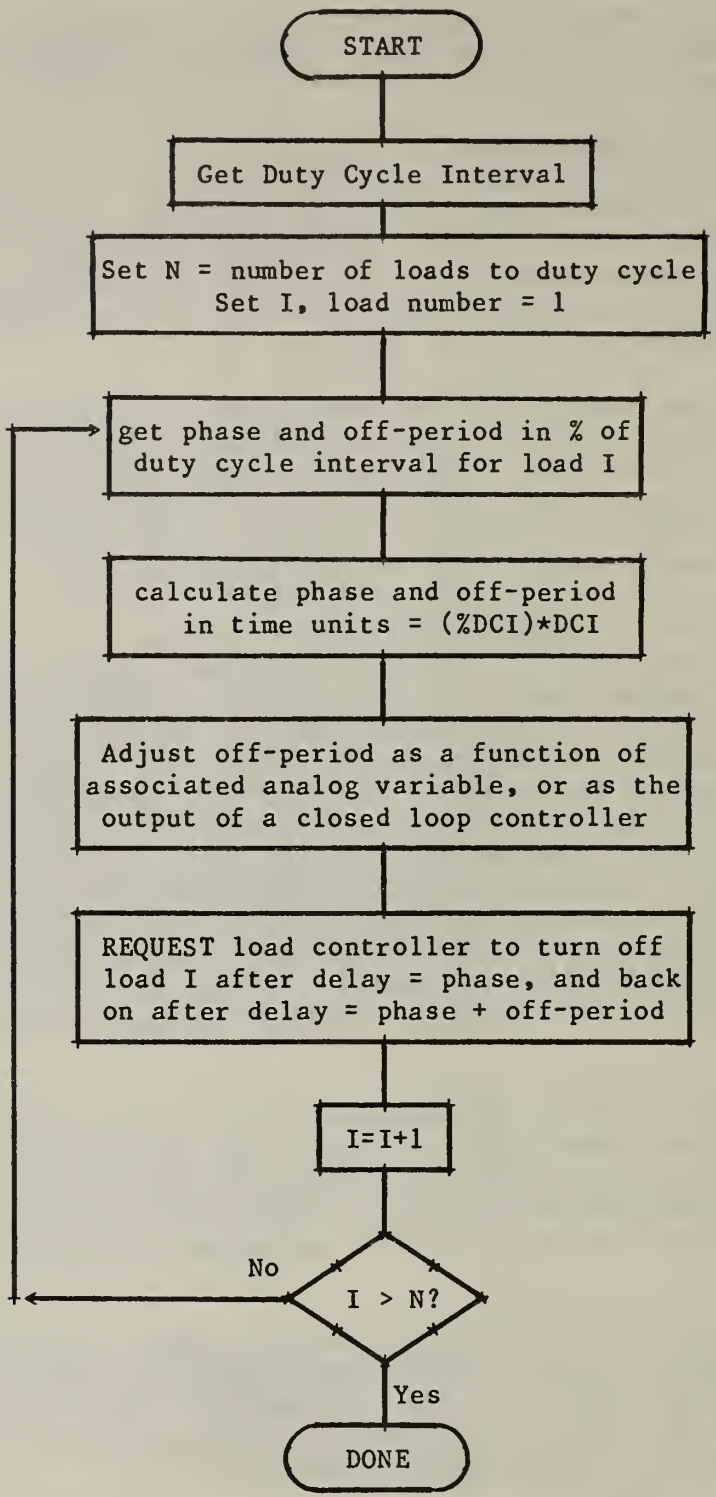


figure 8. duty cycle algorithm incorporating dynamic parameter adjustment

# 4. EXAMPLES OF TIME-OF-DAY AND DUTY CYCLING ALGORITHMS

As part of the research program in building HVAC control at NBS, a prototype BMCS has been constructed using "off-the-shelf" components and microcomputer circuit boards. The system is organized into a hierarchy of three levels: central (CCU), distributed (FID), and data gathering (MUX). Custom software has been developed for the system. The software includes routines for supervisory control as well as direct digital control (DDC). The information in this report is a product of the development effort. Testing of the supervisory control software has been performed using an analog simulator rather than an actual building system. The simulator uses LEDs to represent digital outputs, switches for digital inputs, and potentiometers to represent analog inputs. The time-of-day control software was able to turn LEDs on and off at scheduled times and also initiate more complex events at predetermined times. The duty cycling control software turned LEDs on and off as if they were electrical loads. Tests of the software in an actual building are scheduled to verify operation with a real HVAC system.

Since it is sometimes difficult to construct actual working programs from general algorithms, a description of working time-of-day and duty cycling control software from the NBS FID is included in this chapter. It should be understood that this is one possible solution to implementing these control strategies and is not necessarily the best solution. Other system architectures are certainly possible and will require different implementations of the basic control algorithms.

The software programs presented as examples are intended for operation in the FID level of the BMCS with the exception of the software described in section 4.4 for selecting duty cycle parameters. As background information, section 4.1 will describe the basic architecture of the FID. The sections following 4.1 will describe the examples of software in detail.

## 4.1 Example Field Interface Device Software Architecture

The software to operate the FID is a complex computer program which is mostly written in the high-level computer language, FORTRAN. A number of utility routines are written in microprocessor assembly language, specifically for the Z-80 microprocessor used in the FID.

The activities of the FID can be divided into two major categories: communication with the CCU and execution of tasks. A task is defined as a software procedure that has a specific purpose such as turning on a fan or controlling a valve position and begins at a certain starting point in the FID computer memory. In the NBS FID, a task is in the form of a FORTRAN "subroutine" without arguments. Information is passed between tasks through common data areas. When there is no communication from the central level, tasks are executed with a "multi-tasking" scheme in which a number of software tasks are each executed in a periodic fashion in conjunction with the other tasks so that it appears that all tasks are operating simultaneously. The two

most essential types of tasks of the FID are collection and processing of data and execution of control software (DDC, duty cycling, etc.) through the MUXs.

The time-of-day control software is a task which controls other tasks. The time-of-day control task is concerned with making sure that various functions which have been set up to execute at a specific absolute time-of-day are started on time. Duty cycle software is also a task, which periodically shuts off loads connected to the system to save energy.

Any "multi-tasking" software system must have some sort of "task manager software" to control when tasks will execute and resolve timing conflicts between tasks. The task manager software used for the NBS FID was written in assembly language and makes use of a hardware real-time clock issuing interrupts to divide time into 0.1 second intervals. The task manager uses this division of time to coordinate all tasks in the FID. The FID task manager has three basic characteristics which are desirable for task control. First, the task manager allows the periodic execution of multiple tasks and has a priority scheme for use if two tasks are set to execute at the same time. The period of any task may be changed during execution of the FID software. Second, any periodic task may be stopped on demand. Third, any periodic task which is stopped may be started after a variable delay which is independent of the period of the task, or tasks may be set to execute only once after a variable delay (such a task may be termed a "one-shot" task).

To control tasks, the task manager for the NBS FID uses a table of task starting points and associated time intervals at which these tasks are to be executed. This table will be referred to as the task table. Table 1 is a facsimile of the basic task table in the FID. There is a row in the task table for each task in the FID. There are two groups of columns (four columns per group) used for timing purposes. One group of four columns is designated the counter, and the other group is called the interval. The four columns in a group hold values for hours, minutes, seconds, and tenth-seconds. Every one tenth second, the task manager software executes. The counter for each task

Table 1. NBS FID task table

| task no. | status code | COUNTER hr | min | sec | ts | INTERVAL hr | min | sec | ts | TASK NAME |
|---|---|---|---|---|---|---|---|---|---|---|
| 01 | 1 | 00 | 00 | 60 | 00 | 00 | 00 | 60 | 00 | CONTROL |
| 02 | 1 | 00 | 00 | 01 | 07 | 00 | 00 | 02 | 00 | DATA |
| 03 | 0 | 00 | 05 | 25 | 09 | 00 | 10 | 00 | 00 | INACTIVE |
| 04 | -1 | 00 | 01 | 30 | 00 | 00 | 00 | 00 | 00 | ONESHOT |
| 05 | 1 | 00 | 59 | 59 | 09 | 01 | 00 | 00 | 00 | PERIODIC |
| 06 | 1 | 00 | 00 | 00 | 00 | 00 | 00 | 20 | 00 | ACTIVE |
| 07 | 1 | 00 | 00 | 14 | 02 | 00 | 15 | 00 | 00 | DUTCYC |
| 08 | 1 | 00 | 48 | 08 | 00 | 01 | 00 | 00 | 00 | SCHEDULR |

is decremented each time the task manager is executed. If any counter becomes all zeros, this indicates that the associated task is to be executed. After the task is executed, the counter columns are reset using values from the interval columns, and the countdown resumes. The interval is not changed by normal operation of the task manager.

The first column in the task table contains the task number. This number is also a priority ( a lower number having a higher priority). All tasks are given priorities and lower priority tasks will wait if it is time for a higher priority task to execute. The second column of the task table contains a code to indicate whether the task is stopped (0), set to execute once (-1), or set to execute in a continuous, periodic mode(1). The last column contains a symbolic name which represents the starting address of the task.

A "utility" routine, separate from the task manager routine, is used to change the entries in the task table used by the task manager. This task table editor routine can be called as a subroutine from a FORTRAN program. There are three arguments which must be passed to the routine. The first argument is the number of the task in the table whose table entries are to be edited. The second argument is the value of the code to be loaded into the second column for that row of the table. The third argument is equal to 1 if the task intervals and the task counters for this row in the table are to be set to the same new values. The third argument is set to 2 if only the task counters are to be set to new values. The values to be loaded into the counters and intervals are passed to the subroutine using data in a common block.

Another utility routine is used to determine the current values of the entries in the task table. The routine can be called as a subroutine from a FORTRAN program. There are two arguments. The first argument is the number of the task in the table whose table entries are to be checked. The second entry will contain the value of the code in the second column of the table. The values of the interval columns in the table will be stored in a common block as four consecutive values representing the hours, minutes, seconds, and tenths of seconds. The routine is very similar to the editor routine, but data are "read" from the table rather than being "written" to the table.

### 4.2 Example Time of Day Control Algorithm

As an example of a time-of-day control algorithm, the algorithm in figure 2 was implemented on the NBS FID. The algorithm was developed in FORTRAN as a software task (section 4.1). This task will be referred to as the "scheduler" task. Appendix A contains a FORTRAN listing of the scheduler software.

In this example of time-of-day control, the events which are to be controlled by the scheduler are actually tasks which are existent in the task table (section 4.1). No additional events to be scheduled can be added after the system is in operation since this would involve adding more tasks to the FID

software and lengthening the task table. The scheduler uses a number to refer to tasks which corresponds to the position of the task in the task table. This number is also the priority of the task. The first task is number 1, the second, 2, and so forth.

The information that the scheduler task uses to control events (tasks) is stored in a table in memory. This table, called the schedule table, contains all the information needed to schedule tasks to start or stop at any time of day on any day of the week. A representation of the schedule table is given in table 2. The table is created or changed at the central level by the system. The schedule table may then be loaded into the FID by the CCU software.

There are 5 columns in the schedule table and a number of rows determined by the maximum number of tasks to schedule, for example, 25. Each row is for one task. Column 1 contains the number of the task to be scheduled. Column 2 contains a 1 if the task is to be started as a periodic task, 0 if the task (formerly started as a periodic task)is to be stopped, or -1 if the task is to be executed a single time. Columns 3 and 4 contain the hour and minute of the day (in 24 hr notation) at which the task is to start or stop. The last column is for the day of the week on which the task is to start or stop. The schedule table, as it is loaded from the CCU, is sorted with the task set for the earliest time on the earliest day of the week in the first row. The second row has the task set for the next time on the same day or if no tasks are set for the same day, the earliest task on the next day.

The simplest method of checking the schedule table for tasks to start or stop would be to have the scheduler software execute once per minute. The entire task table could be searched for times matching the current time. If the time

Table 2. NBS FID schedule table

| task no. | control code | TIME OF DAY hour | minute | day of week |
|---|---|---|---|---|
| 10 | 0 | 06 | 15 | 2 |
| 05 | -1 | 09 | 30 | 2 |
| 06 | -1 | 17 | 00 | 2 |
| 05 | -1 | 09 | 30 | 3 |
| 06 | -1 | 17 | 00 | 3 |
| 05 | -1 | 09 | 30 | 4 |
| 27 | 1 | 12 | 00 | 4 |
| 06 | -1 | 17 | 00 | 4 |
| 05 | -1 | 09 | 30 | 5 |
| 06 | -1 | 17 | 00 | 5 |
| 05 | -1 | 09 | 30 | 6 |
| 06 | -1 | 17 | 00 | 6 |

matched, the corresponding task would be stopped or started. In the NBS FID software a somewhat more complex scheme is used to minimize execution time of the scheduler software. The scheduler, rather than executing once per minute, only executes often enough to find all of the tasks which are scheduled to execute. Also, rather than searching the entire schedule table, only that portion containing tasks for the current day is searched (since the table is sorted).

Whenever the FID is reset or started-up or when a new schedule table is loaded from the CCU, the scheduler task is automatically executed. This first execution of the scheduler ensures that the scheduler task will start any tasks that are scheduled to execute within a short time, or if none are set to start, ensures that the scheduler will continue to periodically execute, checking the schedule table.

Figure 9 is a simple flow diagram describing the operation of the scheduler software. The scheduler begins by reading the real time clock to get a day number and determines the current day of the week. The day of the week algorithm is a simple one based on what day of the week the first day of the current year is. The day name for the first day must be entered manually.

After determining the current time of day and day of week, the scheduler goes into the schedule table and positions a pointer at the first entry in the table where the day of the week in the table is equal to or greater than the current day of the week. If the table is empty or the end of the table is reached (signified by a zero for the task ID) the scheduler task is set to re-start 60 minutes later, and an exit is taken from the routine.

The next step for the scheduler task is to determine the time differential in minutes between the current time and the time for the table entry at which the pointer is currently set. Determination of the differential involves a subtraction of the current day and time from the table time and day. Carries are used if the current hour, minute or day is greater than the table hour, minute, or day. If this differential is greater than zero, this means that a task to start or stop is approaching in time. The magnitude of the differential determines what is done next. If the differential is greater than a minimum value (60 minutes) then the scheduler task is set to re-start 60 minutes later, and an exit is taken from the routine. The scheduler task is given a fairly high priority to ensure that it will re-start on time.

If the differential between the table entry time and the current time is less than the minimum value (60 minutes) and the table indicates that the task is to either start for periodic execution or be executed once, the task is caused to start after a time interval equal to the differential (actually, the differential is adjusted so that the task will stop or start as close to the beginning of the minute as possible). This is accomplished by using the task table editor utility routine to change the counter (the current amount of time until task execution) in the task table. The interval (the time between successive executions of the periodic tasks) is not affected.

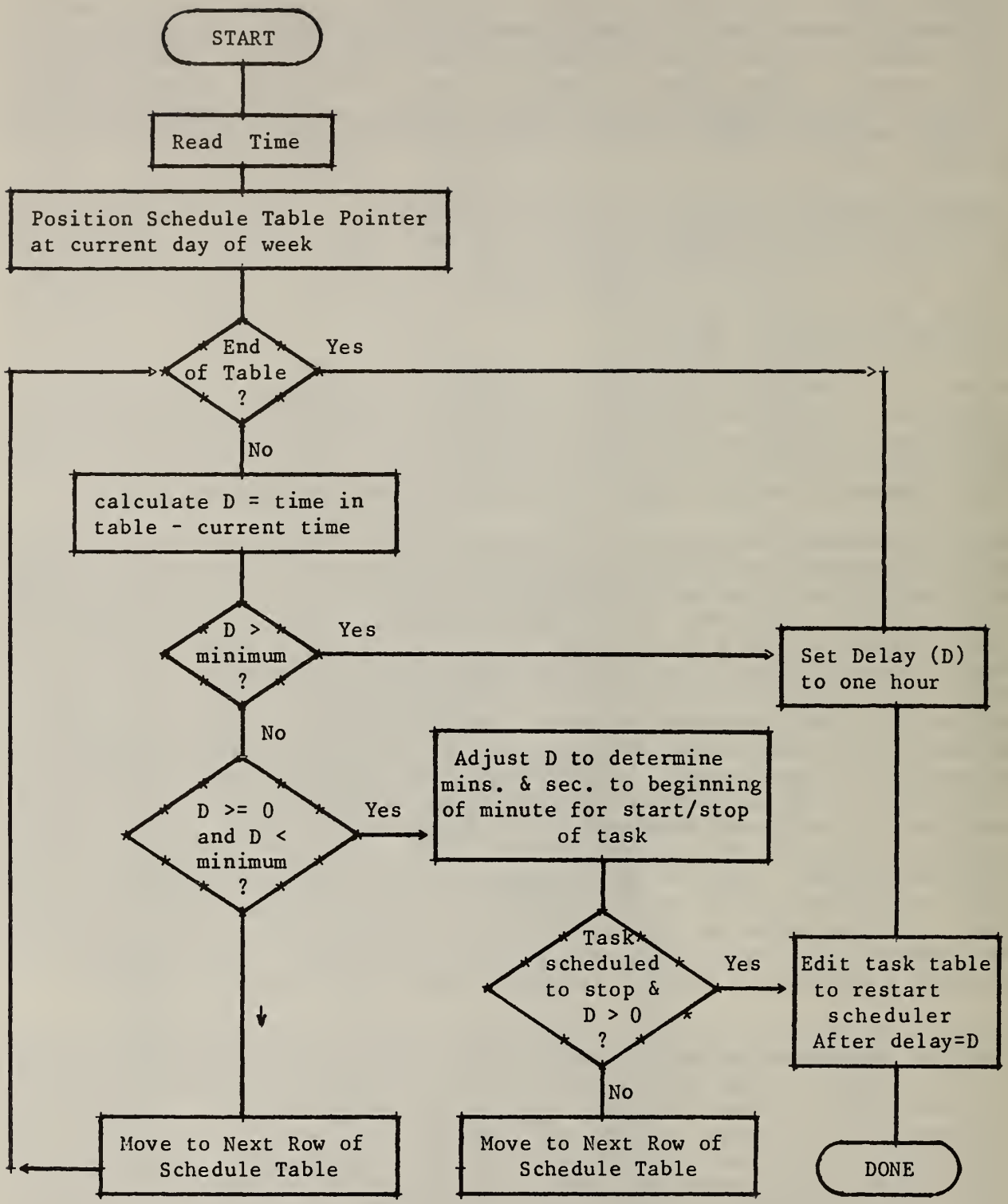


figure 9. algorithm for time-of-day control used in NBS FID

If the differential is less than the minimum, but the task is supposed to be stopped, the task is handled differently than if it were to be started or executed once. The task manager routine has no means of stopping a periodic task after a time interval. Thus the stopping of the task must be done directly by the scheduler at the exact time in the schedule table, rather than indirectly by resetting the counter in the task table when the task execution time draws near. Direct stopping of the task is accomplished by setting the task table counter for the scheduler task to be equal to the adjusted differential as calculated previously. The scheduler task ends, but will re-start in time to stop the task. The task is stopped by checking the differential (which will be zero for the task when the scheduler is re-started) and directly stopping the task if the differential is zero.

If the differential between the table entry time and the current time is less than zero, the task has already been executed. No task control action is taken in this case.

If the schedule table pointer is set at a task which has a differential less than the minimum and which is set to start or execute once, then after the appropriate action is taken to schedule the task, the schedule table pointer is moved to the next table entry. The differential for this entry is then computed to determine if the task for this entry is also to be scheduled for execution. If the differential for this task is small enough and the task is to be started or executed once, then the task table entry for this task is edited, and the the next entry in the schedule table is checked. The schedule table is stepped through until either the end of the table is reached, the differential for a table entry is over the minimum, or a task is to be stopped. Then the scheduler task is set to re-start later, and the scheduler routine ends.

The events listed in the schedule table can be used to cause a number of possible results. An event consisting of a start-up or shutdown of mechanical equipment can be easily implemented by programming the task to turn digital outputs on and off. Sequences of turning digital outputs on can be programmed with delays between starts to reduce high demand loads. Night-time setback of control loop setpoints can be accomplished using a scheduled task to change the value of control loop parameters.

The time-of-day control software worked successfully in the prototype BMCS. In tests, scheduled events always executed within one minute of the scheduled time. The main limitation of the example version of the FID scheduler software is that a task to be executed one time cannot be scheduled to execute more than once in an hour. If the time between executions of a task within an hour is regular, the task can simply be scheduled to start periodic execution.

An obvious improvement to the software would be to add more columns to the schedule table so that the period of a task (the task table interval) could be assigned by the scheduler. Currently, a periodic task can only be started and stopped.

Another possible addition to the software is connected with possible power failures or similar types of system failures. On re-start of the NBS FID, a version of the FID software will be re-loaded from a bubble memory. This version represents the state of the FID system at the time that a configuration save (transfer of RAM to bubble memory) was last performed. If a scheduled load has been turned off or on since the last save, its condition will not be restored. A sixth column could be added to the schedule table to specify a "configuration save" option when the task is started or stopped. If this option were selected, the current FID parameters and selection of outputs in the on state would be saved in non-volatile memory and could be restored in case of system re-start.

A consideration in using the example software is associated with stopping a task. Since a task cannot directly be set to stop after a programmable delay, the scheduler must directly stop the task. If due to some reason the scheduler task is delayed (not likely under normal circumstances since the scheduler task is assigned a high priority) it is possible for the stopping of a task to be missed. If the stopping function is critical, a special task could be used to stop other tasks. This special task could be executed once to stop a periodic task.

## 4.3 Example Duty Cycling Algorithm

Duty cycling control software which is an implementation of the algorithms in figures 6 and 8 was developed for the NBS FID. The algorithm was written in FORTRAN and implemented as a task (section 4.1). Eight support routines were written in FORTRAN and made into tasks for a total of nine tasks related to duty cycling. Appendix B contains the source code listings for the duty cycling software.

The duty cycle parameters for the loads to be duty cycled are kept in a table in the NBS FID referred to as the duty cycle table. A representation of part of the duty cycle table is given in table 3. The values of the phases and off-times in the duty cycling table are initially determined by software at the central level for a set of loads and constraints. The table is then transmitted to the FID. The FID then uses the off-period values from the CCU as maximum design values (section 3.6).

table 3. NBS FID duty cycle table

| LOAD no. | | phase | off-period | | | | inter- |
|---|---|---|---|---|---|---|---|
| mux | pnt | (%DCI) | (%DCI) | min-off | min-on | cycle? | lock |
| 01 | 25 | 00 | 30 | 1 min | 2 min | YES | 0 |
| 02 | 05 | 20 | 25 | 1 min | 1 min | NO | 0 |
| 01 | 19 | 30 | 50 | 2 min | 1 min | YES | 0 |
| 01 | 02 | 50 | 10 | 1 min | 1 min | YES | 0 |

The duty cycling task has a row in the FID task table (table 1), and the task table interval columns are adjusted to cause the task to execute at a frequency which is the reciprocal of the duty cycle interval. The duty cycle task is in the form of a FORTRAN subroutine which uses a heirarchy of four lower levels of subroutines to execute the duty cycling functions. Each of the levels calls a lower level with more primitive functions than the calling routine. Figure 10 is a diagram of the duty cycling routines. The main routine is shown at the upper left of the diagram with three levels of subroutines displayed around it.

The main duty cycling routine reads parameters directly from the duty cycle table. Each row of the table contains the digital output number representing the load to cycle, the phase and off-time parameters, and a flag to indicate whether or not the load should be cycled. The phase and off-time parameters are expressed in terms of a percentage of the duty cycle interval. For example, if the off-time is to be 5 minutes and the duty cycle interval is 20 minutes, then the off-time percentage is 25 percent. The duty cycle main routine must convert the percentages into absolute times. To do this the duty cycle interval must be known. A utility routine is used to determine the duty cycle interval in the task table (table 1).

In the NBS duty cycling software a method of adjusting the duty cycle table based on the technique of section 3.6 was implemented. It is assummed that the off-periods in the duty cycle table (table 3) are maximums, determined for a design point, and that the off-times are to be modulated downward as a function of an associated analog value (outside air temperature, ventilation, space conditioning requirements, occupancy, etc.). The points at which the duty cycling off-period reaches zero are specified by low limit and high limit values of the analog value. It is assummed that the magnitude of the associated analog value at the design point lies between these two limits (see figure 7).

The actual implementation of the duty cycle table adjustment in the NBS FID uses an algorithm to modify the duty cycling phase and off-period values, taken from the table for a particular load, before the duty cycling task requests the load controller to turn the load off and back on. For each load, four additional table entries must be specified, in addition to the columns shown in table 3.

Table 4 shows the additional entries to the duty cycle table required for the adjustment of duty cycle off-period. An analog sensor point to be used as the basis for adjusting the off-period must be specified by point number. This could be a measurement such as outside air temperature or a calculated value such as air handler load. Also the three analog parameters illustrated in figure 7, the design point, low end point, and high end point, must be specifed. These quantities must be entered in the units for the associated analog value (such as degrees C for outside air temperature).

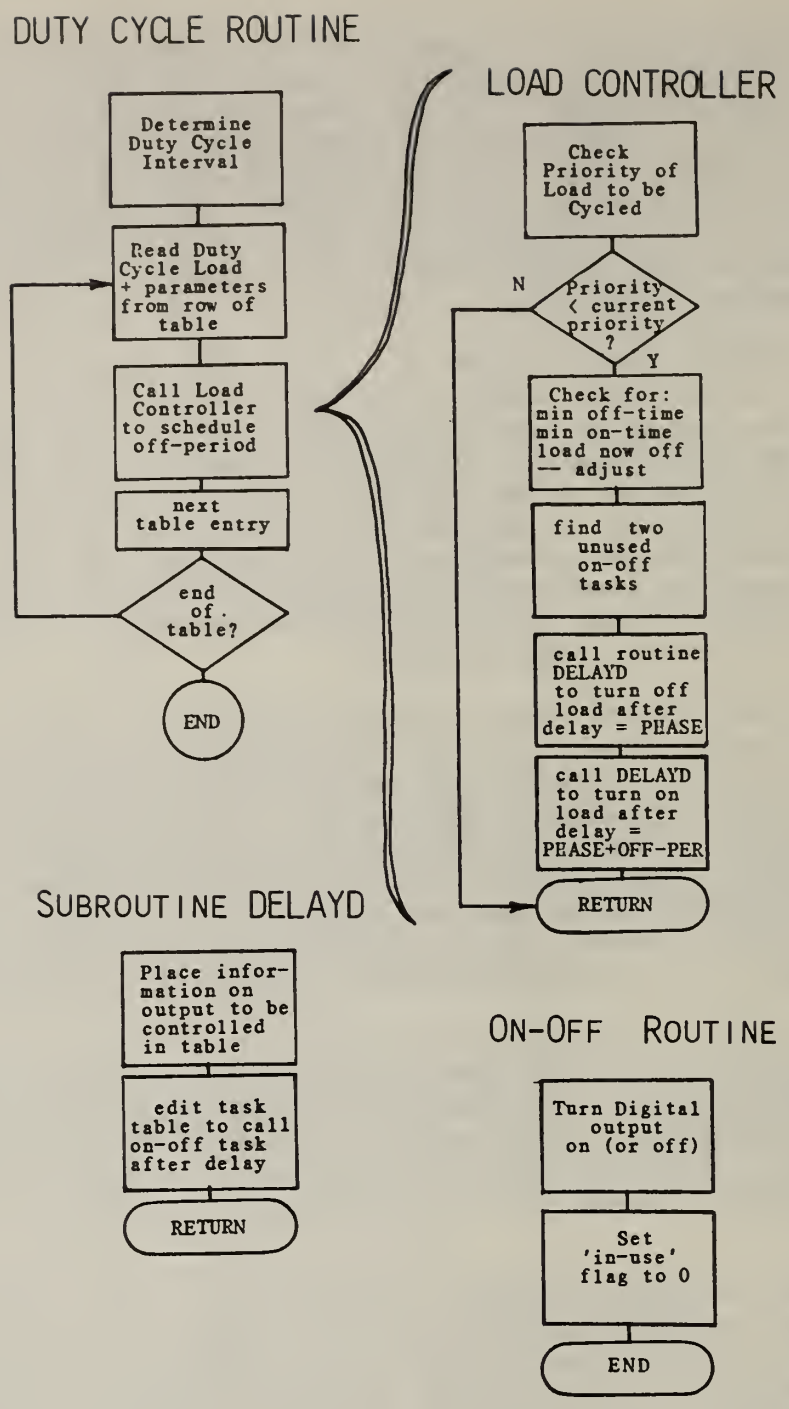


figure 10. flow diagrams for example duty cycling task

Table 4. Additions to duty cycle table for dynamic adjustment

| LOAD no. mux | pnt | inter-lock | associated analog mux | pnt | design analog | lo-end analog | hi-end analog |
|---|---|---|---|---|---|---|---|
| 01 | 25 | 1 | 01 | 32 | 15.5 | 00.0 | 25.0 |
| 02 | 05 | 0 | 02 | 16 | 25.0 | -10.0 | 35.0 |
| 01 | 19 | 1 | 01 | 02 | 25.0 | 20.0 | 30.0 |
| 01 | 02 | 1 | 01 | 02 | 20.0 | 05.0 | 20.0 |

The algorithm to modify the duty cycle off-period uses the values shown in table 4 to calculate the proper off-period and phase values. These calculations are best represented by equations. The first step is to calculate the difference between the current value of the associated analog variable and the design point value of this variable:

$$D = (\text{design point}) - (\text{analog value}) \quad . \tag{1}$$

The next step is to calculate a value for the fractional reduction of the maximum duty cycle off-time to be used at this level of the associated analog value. If D is less than zero (right side of figure 7), then the fraction is defined by:

$$F = \frac{D}{(\text{design point}) - (\text{high end point})} \quad . \tag{2}$$

However, if D is greater than zero, then the following equation is used:

$$F = \frac{D}{(\text{design point}) - (\text{low end point})} \quad . \tag{3}$$

The reduction in the duty cycle off-period is then given by:

$$R = (\text{absolute maximum off-period}) * F \tag{4}$$

where 'absolute maximum off-period' is the design off-period value in minutes. The absolute off-period is that portion of the duty cycle interval specified by the design off-period in percent, taken from the duty cycle table for the load currently being duty cycled.

The absolute phase, in minutes, to be used for this value of the associated analog is calculated by adding R to the portion of the duty cycle interval specified by the the phase entry in the duty cycle table (in percent). The absolute off-period for this value of the associated analog is obtained by

subracting R from the absolute maximum off-period. Note that this method results in the time the load is turned back on remaining the same, since:

$$\begin{aligned}(\text{on-time}) &= [\text{adjusted phase}] + [\text{adjusted off-period}] \\ &= [(\text{design phase}) + R] + [(\text{design off-period}) - R] \\ &= (\text{design phase}) + (\text{design off-period}) \quad .\end{aligned} \tag{5}$$

Therefore, no algorithm is required to coordinate load turn-on times after each adjustment of parameters. After the final values of phase and off-period have been calculated, the main duty cycle routine then calls a subroutine, the "load controller", to actually turn the load on and off.

The function of the load controller is to allow the main duty cycle routine to request the deactivation of a given load for a specific time period (off-period) after a specified delay (phase). The load controller subroutine has five arguments. These are the load number, the phase (in minutes and seconds), the off-period (minutes and seconds), a load control priority which is arbitrarily specified as level five for the duty cycler, and a status variable. The status variable returns the status of the load control function. A zero indicates that the load control request was implemented successfully. If the status variable is a one, this indicates that the request for load control was rejected because the load is currently under control by a higher priority function. Advanced versions of the FID might have more status variable assignments and might take various actions on receipt of certain statuses. After taking control actions for all of the loads entered in the table, the duty cycler task ends. After a time period equal to the duty cycle interval, the duty cycler task starts again.

The load controller is a general purpose routine that can be called by any task or function which must request the turning off of a digital output for a certain time after a delay. The routine would be used in particular by a demand limiting function if it were implemented in the FID software. The load controller routine does the following things: checks to see if the load which is to be controlled is currently under control by a routine with higher priority than the routine requesting control; determines if minimum off-period criteria are satisfied; verifies that minimum on-period criteria are satisfied; and calls the appropriate subroutines to cause the desired load to turn off and then on, if all criteria are met.

To implement the concept of priority control of digital outputs, a priority variable is used for each output. This number describes the priority level at which the output is currently controlled. The higher the number, the lower the priority. When a digital output is actuated, the routine setting the output fixes the priority so that lower priority functions cannot take control of the output.

The duty cycler software makes use of a battery of small nearly identical routines, called "on-off" routines, which are actually tasks entered in the task table. The function of any of the on-off routines is to turn off or on a

digital output. Any on-off routine can control any load. Information on the output that is to be controlled, whether it is to be turned on or off, and the control priority, is located in a software data table which all of the on-off routines can access. Each on-off routine has a row in the table which always contains data just for that routine. The entries in the table may be changed, making the on-off routine programmable. When an on-off routine executes, it checks values in the table to determine the output to control. On-off tasks (each with a unique task number) are set to execute by the duty cycling software using the task table editor. Once a routine has been set to execute at a future time, an entry in the table is used to mark that routine as in-use.

Since a given on-off routine may have been previously assigned an in-use condition, the load controller routine must use an algorithm to select unused on-off routines before a load can be programmed for future start or stop. For each load that must be turned off and then on again in a duty cycle interval, two on-off routines are needed. One routine must turn off a duty cycle load, and the other must turn the load back on. The load controller selects two unused on-off routines by checking these in-use entries in the on-off routine table. Then, to turn off the load to be duty cycled, the load controller calls a subroutine (which directly controls the on-off tasks) to turn the load off after a delay equal to the phase for that load. A second call is made to the same subroutine to turn the load back on after a delay equal to the phase plus the off-period.

Six arguments must be passed to the subroutine used to control the on-off routines. They are: the identifier number of an on-off routine to use, a digital output number, the priority used to control the digital output, a 1 or 0 to indicate if the load is to be turned off or on, and the number of minutes and seconds before the digital output is to be controlled. The programmable delay control routine sets appropriate values in the on-off routine table and then calls the task editor to set the task table counter for the selected on-off routine to be equal to the delay passed as an argument from the load controller.

When the on-off tasks execute, they check the table for the correct digital output to control and then make use of a primitive digital output control routine to physically control the digital output. After setting the output, the in-use flag for that particular on-off routine is reset to zero, thus allowing the routine to be used for control of another digital output.

The digital output control subroutine is also used by closed loop controller tasks to turn on and off digital outputs used for damper and valve positioning. There are three arguments passed: the output number, a 0 or 1 to indicate off or on, and a priority value. Using the MUX commands and the FID-MUX communication protocols, the digital output control routine then commands the MUX to physically control the output. The digital output to be set is passed as a two part variable which is made up from the MUX number (first part) and the digital output number within that MUX (second part).

The digital output control subroutine has a special feature which provides a solution to a problem which can arise in the operation of the duty cycler. The problem can occur when an output which is currently being duty cycled is turned off with a high priority by a task such as the scheduler (for example, the scheduler turns off an air handling unit fan at the end of a day). If the output is turned off by a high priority function when no on-off task has been programmed to control that output by the duty cycler (a situation that might exist near the end of a duty cycle interval for a load with a small phase and a small off-time), the output will be turned off and a high priority will be entered into the digital output priority variable for that output. When the duty cycle interval begins again, the duty cycler finds that the output is under high priority control and the output is not controlled. However, if the output is turned off by a high priority function when on-off tasks have been programmed to control the output (such as for a load with a short phase and a long off time, near the beginning of a duty cycle interval), then when the on-off tasks turn the output off and then on again, the high priority value of the flag for the digital output will be over-written. This occurs because the digital output control routine does not normally compare new priorities with old priorities but unconditionally sets the output. If this were not possible, no digital output priority could be set to a lower value once set to a high value.

The duty cycle problem is solved by having the digital output control routine check priorities when called by the on-off routines turning on an output. In this case, the absolute value of the new priority for the digital output is compared with the old priority, and the output will not be set if the new priority is lower than the old priority.

The duty cycling software was tested on the NBS laboratory BMCS and used to duty cycle loads on the analog simulator. A group of four LEDs was used to represent electrical loads. The duty cycling software successfully cycled the loads. In addition, loads were not cycled when the scheduler (time-of-day control) software caused the loads to be turned off at a particular time.

4. Example Duty Cycle Parameter Selection

A general algorithm was developed in FORTRAN to create a duty cycle table as mentioned in section 3.5. This algorithm, intended to run at the CCU level, will be referred to as the duty cycling phase allotment algorithm. The CCU computer operator must specify the loads to be duty cycled and the desired percentage of the duty cycle interval that the load is to be in the off state at design conditions. In addition, the operator may specify relationships to be maintained between loads. These load relationships may be either in the form of interlocking (loads must be off at the same time and on at the same time) or in the form of exclusivity (two loads must not be on or off at the same time). The inclusion of relationship specification makes the algorithm reasonable complex. The algorithm developed makes use of a "brute force" technique to move load off-periods around within the duty cycle interval (effectively by allotting phases to the loads) until all criteria are

satisfied. The main criterion remains the avoidance of simultaneous activations of loads. A brief description of the algorithm will be given here. Appendix C contains a more detailed description and the source code listing of the example program.

A flow chart of the general operation of the phase allotment algorithm is given in figure 11. The algorithm works on a duty cycle table such as shown in table 3. All off-times and phases are specified in terms of a percentage of the duty cycle interval. Five major passes are made through the table to satisfy all of the constraints. In figure 11, only the first two passes are shown in detail. Additional figures in appendix C illustrate the other passes. The first two passes are sufficient to set up a table for a simple situation where the loads in the table have fairly small off-times and no load relationships are specified. The other passes are used to sort out more complicated situations.

The guiding philosophy for the first two passes of the phase allotment algorithm is to arrange the off periods of the loads within the duty cycle interval so that the on-times of the loads are as evenly spaced throughout the interval as possible. There are usually a large number of solutions to this problem, but one particular arrangement is used. The first pass divides the duty cycle interval into equal intervals, identifying the same number of ideal load turn-on times as there are loads to duty cycle. The second pass then attempts to assign phases to the loads so that the loads are turned on at the ideal turn-on times. Loads which cannot be assigned ideal turn-on times are flagged to be taken care of in subsequent passes.

The third, fourth, and fifth passes of the phase allotment algorithm shown in figure 11 modify the basic duty cycling table for complex duty cycling constraints. The third pass assigns phases to loads which could not be forced into turning on a load at an ideal turn-on time determined in the first pass of the algorithm. The fourth pass adjusts the table for interlocking between loads, and the fifth pass is used to satisfy mutually exclusive load constraints. If any of these types of constraints are absent, the third, fourth, or fifth passes may be skipped.

The third pass of the algorithm operates by moving load off-periods around within the duty cycle interval. If a load which could not be assigned an ideal turn on time is encountered, it is tested to determine how close its turn-on time would be to the nearest other turn-on time if the load were given a phase of zero. If too close, the load off-period is moved until there is sufficient space between load turn-on times.

The fourth pass of the phase allotment algorithm consists of checks of the duty cycling table for entries in the column for interlocks between loads. If the load being checked is interlocked to another load, the off-period for the load is moved within the duty cycle interval, and a compromise is made between having the two loads being off at the same time and avoiding the situation where two loads have turn-on times that are too close together. When all

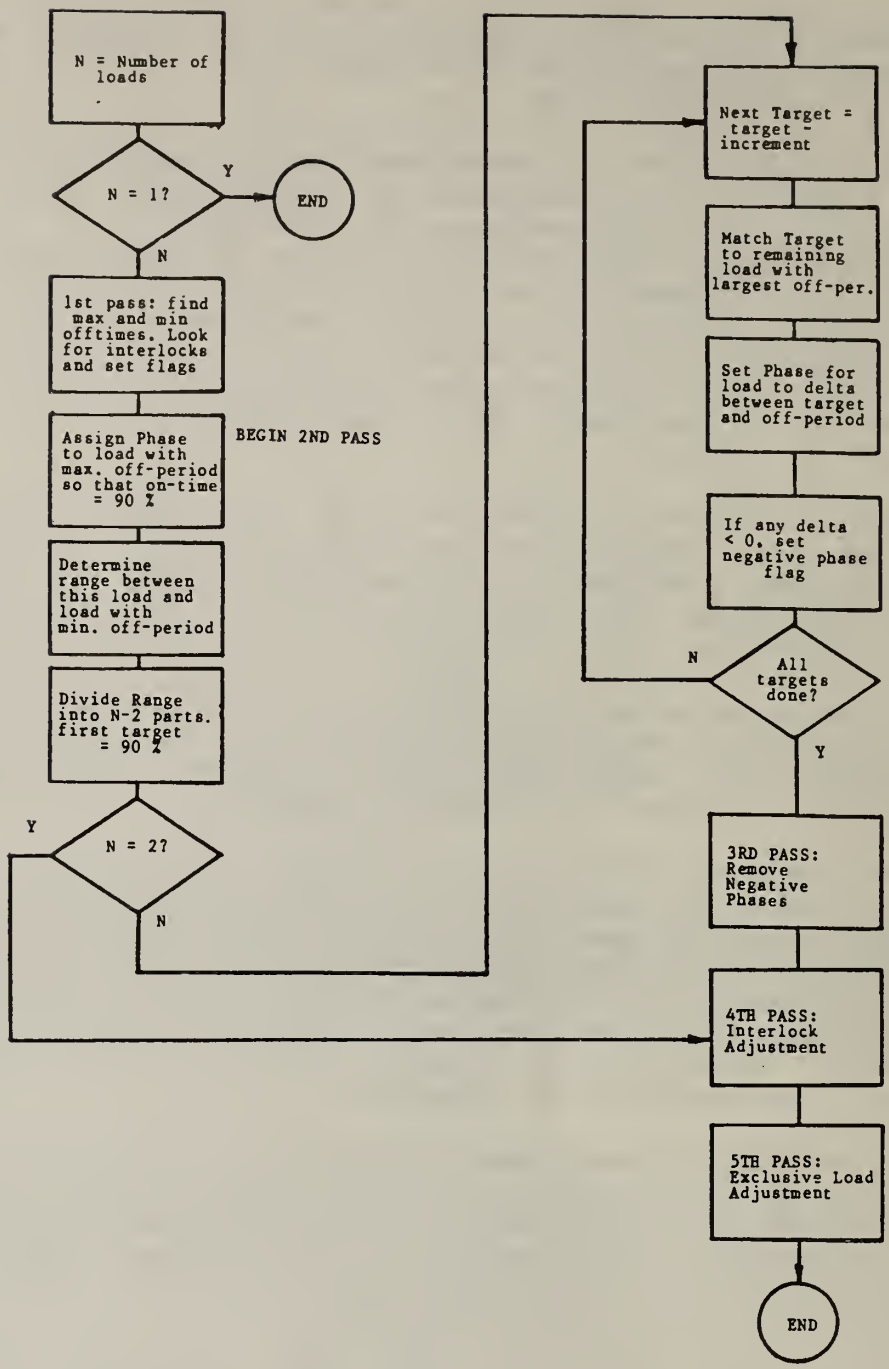


figure 11. Flow diagrams for phase allotment algorithm

loads have been checked, the fourth pass is complete and all loads should be turned off at approximately the same time as any loads with which they are interlocked.

The fifth and final pass of the phase allotment algorithm adjusts the table for loads which should not be off at the same time as another load which is specified. As in the fourth pass, load off-periods are moved within the duty cycle interval, if required. In this case, however, load off-periods are moved so that they do not coincide with the off-period of a specified load, while maintaining a minimum distance between load on-times.The fifth pass can become complicated if there are multiple interlocks between loads that conflict with each other.

When the fifth pass is complete, the duty cycle table is finished and should reasonably satisfy all constraints. The final table may not be the best solution to the problem because the algorithm is relatively simple. The operator may wish to make slight improvements to the table, after it has been created by the phase allotment algorithm.

## 5. CONCLUSIONS

This report has described algorithms developed for use in the NBS Building Management and Controls Laboratory for two supervisory building control strategies. The strategies implemented were time-of-day control and duty cycling. The algorithms were tested using an actual microprocessor-based FID running software incorporating the algorithms. The algorithms developed performed successfully on a test system using a simulator panel rather than an actual building system. Further research will be required to determine how the algorithms perform when connected to a real HVAC system and to measure the performance of the algorithms in terms of energy savings, accuracy of control, and reliability.

## 6. REFERENCES


1. U.S. Army Corps of Engineers, "Energy Monitoring and Control Systems," Technical Manual TM5-815-2/AFM 88-36/NAVFAC DM-4.9, Dept. of the Army, Air Force, and Navy, Washington, DC, September 1, 1981.

2. "Duty Cycling's extra heat: will it void warranty?," Energy User News, 4:7 1,10-11p. 12 Feb, 1979.

## APPENDIX A - Time-of-Day Control Routines

This appendix contains the FORTRAN IV source code for the time-of-day control routines used in the NBS FID. The routines are taken out of the complete program that they are designed to work with, but still provide an example of actual source code for time-of-day control.

```
C::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
      SUBROUTINE SCHTSK
C::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
C This routine is the task which is responsible for time of day control.
C Information from the CCU level has been loaded into a schedule table
C and this table is used by this scheduler task to cause the starting
C and stopping of tasks.
C
C VARIABLE DEFINITIONS:
C
C COMMON BLOCK SCHEDU: the schedule table; values entered by other tasks
C TASKID - array of task ID numbers representing scheduled events
C ISSO   - array of values to indicate stop, start, or one-shot task
C THR    - array of hours of day to start tasks
C TMIN   - array of minutes within the hour to start tasks
C TDOW   - array of days of the week to start tasks
C
C COMMON BLOCK TSKLNK: parameters describing the tasks in the FID software
C TSKMAX - total number of tasks currently configured in the system
C
C KS     - index of starting position (row) in schedule table
C NCD    - day number of the current day of the year (1-365)
C NCDOW  - number representing the current day of the week (1-7)
C NCH    - number of the current hour of the day
C NCM    - number of the current minute of the hour
C NDIFF  - number of differential hours between table and current time
C NDIFM  - number of differential minutes between table and current time
C NTD    - day of week that the current task is scheduled to execute
C NTH    - hour that the current task is scheduled to execute
C MINMUM - minimum time differential for immediate task scheduling
C TIME   - array containing current time, obtained from hardware clock
C
      BYTE TASKID,ISSO,THR,TMIN,TDOW,TIME(9),NCDOW,H,M,S,TS
      BYTE CNTRL,IDOFF,SCAN0,SCMAST,SCSLAV,DCSLAV,TSKMAX,DCMAST
      COMMON/TSKLNK/CNTRL(16),IDOFF(16),SCAN0,SCMAST,SCSLAV(16),
     &DCSLAV(8),TSKMAX,DCMAST
      COMMON/SCHEDU/TASKID(25),ISSO(25),THR(25),TMIN(25),TDOW(25),KTASK
      COMMON/TSKINT/H,M,S,TS
      DATA MINMUM/60/
C----------determine current time and day of week----------------------
      CALL RDCLOK(TIME)
```

```
      NCD=TIME(9)*100+TIME(8)*10+TIME(7)
      NCDOW=IDOW(NCD)
      NCH=TIME(6)*10+TIME(5)
      NCM=TIME(4)*10+TIME(3)
C-----------find starting point in task table--------------------------
      IF(TASKID(1).EQ.0)GO TO 9999
         DO 200 K=1,24
         IF(TASKID(K).EQ.0.OR.TDOW(K).GE.NCDOW)GO TO 1000
  200    CONTINUE
C-----------------check nearness of tasks in table---------------------
 1000 KS=K
         DO 1200 K=KS,24
C-------------time differential calculation----------------------------
         IF(TASKID(K).EQ.0)GO TO 8000
         NCARRY=0
         NTH=THR(K)
         NTD=TDOW(K)
         NDIFM=TMIN(K)-NCM
         IF(NDIFM.GE.0)GO TO 1040
         NDIFM=NDIFM+60
         NTH=NTH-1
 1040    NDIFF=NTH-NCH
         IF(NDIFF.GE.0)GO TO 1060
         NDIFF=NDIFF+24
         NTD=NTD-1
 1060    NDIFM=NDIFM+(NDIFF*60)
         NDIFF=NTD-NCDOW
         IF(NDIFF.GE.0)GO TO 1100
         NDIFF=NDIFF+7
         NCARRY=-1
 1100    NDIFM=NDIFM+(NDIFF*1440)
         IF(NCARRY.EQ.-1)NDIFM=NDIFM-10080
C--------------based on differential, shedule task or quit--------------
         IF(NDIFM.LT.MINMUM.AND.NDIFM.GE.0)GO TO 2000
         IF(NDIFM.GE.MINMUM)GO TO 8000
 1200    CONTINUE
      GO TO 8000
C-----------------schedule a task for execution-----------------------
 2000 TS=0
      H=0
      M=NDIFM-1
      IF(NDIFM.LE.0)M=0
      S=60-(TIME(2)*10+TIME(1))
      IF(NDIFM.LE.0)S=0
      IF(ISSO(K).EQ.0.AND.NDIFM.GT.0)GO TO 8500
      IF(TASKID(K).LE.TSKMAX)CALL TSKEDT(TASKID(K),ISSO(K),2)
      GO TO 1200
C---------if no tasks to schedule, set scheduler task to restart later--
 8000 TS=0
```

```
      H=1
      M=0
      S=0
 8500 CALL TSKEDT(T,1,1)
C----------------exit----------------------------------------------------
 9999 RETURN
      END

C=================================================================
      FUNCTION IDOW(IDAY)
C=================================================================
C Determines the day of the week as a function of the day number
C for 1983 only. Sunday = 1, Mon. = 2, etc.
C IDOW = day of the week, NWEEK = week number, IDAY = day number
      DATA IDOW1/7/
      NWEEK=IDAY/7
      IDOW=IDAY-(NWEEK*7)+IDOW1-1
      IF(IDOW.GT.7)IDOW=IDOW-7
      RETURN
      END
```

## APPENDIX B - Duty Cycling Control Routines

This appendix contains the FORTRAN IV source code for the duty cycling control routines used in the NBS FID. The routines are taken out of the complete program that they are designed to work with, but still provide an example of actual source code for duty cycling control.

```
C::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
      SUBROUTINE DUTCYC
C::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
C This routine is the main routine to cause duty cycling of electrical
C loads connected to digital outputs to occur. Based on the contents
C of a duty cycle table, the routine causes digital outputs currently
C on to be turned off for certain periods of time to save energy.
C The routine LDCONT is used to create an interface to the loads.
C
C VARIABLE DEFINITIONS:
C
C COMMON BLOCK DUTYCT: The duty cycle table. It is loaded from the CCU.
C LOAD   - array of load ID numbers to be duty cycled (MUX no. and point no.)
C PCPHAS - array of phase times for the loads (% of duty cycle interval)
C PCOFF  - array of off-period times for the loads(% of duty cycle interval)
C ADJUST - logical array, 1 if off-period is to be dynamically adjusted
C DCAMUX - array of MUX ID numbers for the analog values used for adjustment
C DCAPNT - array of Point ID's for the analog values used for adjustment
C DCADES - array of values for the adjustment analog value at the design point
C DCALO  - array of lower values of the adjustment analog at minimum off-period
C DCAHI  - array of higher values of the adjustment analog at min. off-period

C DCMAST  - task ID number for duty cycler task (passed in common block)
C DCI     - duty cycle interval obtained from task manager task table
C DELTA   - design minus current analog value for off-period adjustment
C FRAC    - difference between design and high (low) adjustment analog values
C H,M,S,TS- task execution interval times from task table (hrs., mins., etc.)
C IEDOS   - argument for task interval utility routine, holds task status
C PHASE   - two part integer containing phase for current load in mins. and sec
C REDUC   - reduction in off-period when duty cycle adjustment is made
C STATUS  - status of request to load controller to turn load off and on
C TIMOFF  - two part integer containing off-period for current load (min.,sec.)
C TPHASE  - absolute phase in seconds for current load
C TTO     - absolute off-period in seconds for current load
C
      INTEGER LOAD,PCPHAS,PCOFF,DCI,TTO,TPHASE
      BYTE ADJUST,DCAMUX,DCAPNT
      REAL DCADES,DCALO,DCAHI
      REAL*8 ANAI
      REAL DELTA,FRAC
```

```
      INTEGER REDUC,LOAD,PHASE(2),TIMOFF(2),STATUS
      BYTE CNTRL,IDOFF,SCAN0,SCMAST,SCSLAV,DCSLAV,TSKMAX,DCMAST
      BYTE H,M,S,TS
      COMMON/TSKINT/H,M,S,TS
      COMMON/TSKLNK/CNTRL(16),IDOFF(16),SCAN0,SCMAST,SCSLAV(16),
     &DCSLAV(8),TSKMAX,DCMAST
      COMMON/DUTYCT/LOAD(16),PCPHAS(16),PCOFF(16),ADJUST(16),DCAMUX(16),
     &DCAPNT(16),DCADES(16),DCALO(16),DCAHI(16)
      COMMON/ANALOG/ANAI(1,32)
C
C-------------determine duty cycle interval DCI-------------------------
      CALL TSKCHK(DCMAST,IEDOS)
      DCI=M*60+S
C----------read duty cycle loads and parameters from table--------------
         DO 1000 I=1,16
         IF(LOAD(I).LE.0)GO TO 1000
         TTO=DCI*PCOFF(I)/100
         TPHASE=DCI*PCPHAS(I)/100
         IF(ADJUST(I).EQ.0)GO TO 500
C----------adjust PCOFF and PCPHAS if required-------------------------
         M=DCAMUX(I)
         S=DCAPNT(I)
         DELTA=DCADES(I)-ANAI(M,S)
         IF(DELTA.LE.0)FRAC=DCADES(I)-DCAHI(I)
         IF(DELTA.GT.0)FRAC=DCADES(I)-DCALO(I)
         IF(FRAC.EQ.0)GO TO 500
         FRAC=DELTA/FRAC
         REDUC=TTO*FRAC
         TPHASE=TPHASE+REDUC
         TTO=TTO-REDUC
C----------------Call load controller to turn load on and off-----------
  500    TIMOFF(1)=TTO/60
         TIMOFF(2)=TTO-TIMOFF(1)*60
         PHASE(1)=TPHASE/60
         PHASE(2)=TPHASE-PHASE(1)*60
         CALL LDCONT(LOAD(I),PHASE,TIMOFF,5,STATUS)
 1000    CONTINUE
C------------Check status and take appropriate action------------------
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C In this version, no status check is made. Status check becomes       C
C important when demand limit control is added to the FID.            C
C If records of duty cycling are to be kept, this routine must also    C
C keep track of the actual time that loads are cycled off             C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      RETURN
      END
C=====================================================================
      SUBROUTINE LDCONT(LOAD,PHASE,TIMOFF,PRIOR,STATUS)
C=====================================================================
```

```
C This is the load controller routine. It is used by routines which
C control electrical loads controlled by digital outputs. The routine
C performs the following functions. 1: checks to see if the load is
C currently under control by a routine with a higher priority than
C the routine requesting control; 2: Checks to see if minimum off-time
C criteria are satisfied; 3: Checks for a violation of minimum on-time
C criteria; 4: controls the load by the use of two currently unused
C floating on-off control tasks.
C
C VARIABLE DEFINITIONS:

C CURPRI - current priority level that digital output to be controlled is at
C DOPRI  - array containing current control priority of all digital outputs
C INUSE  - logical array indicating if an on-off task is reserved
C LOAD   - ID number of the load to be cycled off
C MINOFT - minimum value of the off-period to avoid equipment damage
C NEGPRI - negative of PRIOR, used to turn on load and request a priority check
C ONAGIN - relative time interval from present before load is turned back on
C PHASE  - relative time interval from present before load is turned off
C PRIOR  - priority that load is to be controlled under
C STATUS - status of request for load control:
C   status = 0 , loads successfully controlled
C   status = 1 , load control rejected due to low priority request
C   status = 2 , off-time less than minimum specification
C TIMOFF - off-period for load
C
      BYTE DIGO,DOPRI,DIGIN
      BYTE DONOFF,INUSE
      INTEGER LOAD,PHASE(2),TIMOFF(2),PRIOR,STATUS,MINOFT
      INTEGER ONAGIN(2),IPR,DNUM
      INTEGER LOAD2,CURPRI,NEGPRI
      BYTE MUXPNT(2)
      COMMON/ONOFF/DNUM(8),DONOFF(8),INUSE(8),IPR(8)
      COMMON/DIGITA/DIGO(1,24),DOPRI(1,24),DIGIN(1,16)
      EQUIVALENCE (LOAD2,MUXPNT(1))
      DATA MINOFT/1/
C-----------------priority check---------------------------------------
      LOAD2=LOAD
      I2=MUXPNT(1)
      I1=MUXPNT(2)
      CURPRI=DOPRI(I1,I2)
      IF(PRIOR.LE.CURPRI)GO TO 100
      STATUS=1
      RETURN
C---------minimum on-time check---------------------------------------
  100 CONTINUE
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C Minimum on-time checking is not implemented in this version. When    C
C demand limit control is added to the FID, this routine must check to C
```

```
C see if the load to be turned off has been on long enough to avoid    C
C damage to equipment or satisfy other criteria. If demand limit is    C
C trying to turn off a load, the duty cycler may just have turned a    C
C load on. A minimum on time must elapse, so the time until the load   C
C is turned off, the phase, must be adjusted. If duty cycle is trying  C
C to turn off a load, and demand limit has just released a load, then  C
C there must also be a minimum on time, and the phase must be adjusted.C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C--------minimum off-time check-----------------------------------------
C Minimum off time becomes important when the percent off time becomes C
C too small and there is the risk of equipment damage as equipment is  C
C turned off and then on again a short time later. Currently, minimum  C
C is set at 1 second.                                                  C
C-----------------------------------------------------------------------
      IF(TIMOFF(1).GT.0.OR.TIMOFF(2).GT.MINOFT)GO TO 200
      STATUS=2
      RETURN
C----------find unused on/off task pair--------------------------------
  200 I1=0
      I2=0
         DO 1000 I=1,8
         IF(INUSE(I).NE.0)GO TO 1000
         IF(I1.NE.0)I2=I
         IF(I1.EQ.0)INUSE(I)=1
         IF(I1.EQ.0)I1=I
         IF(I1.NE.0.AND.I2.NE.0)GO TO 2000
 1000    CONTINUE
      I1=5
      I2=6
 2000 INUSE(I2)=1
C----------set load to turn off and then on-----------------------------
      NEGPRI=-PRIOR
      CALL DELAYD(I1,LOAD,NEGPRI,0,PHASE(1),PHASE(2))
      ONAGIN(1)=PHASE(1)+TIMOFF(1)
      ONAGIN(2)=PHASE(2)+TIMOFF(2)
      CALL DELAYD(I2,LOAD,NEGPRI,1,ONAGIN(1),ONAGIN(2))
      STATUS=0
      RETURN
      END
C=======================================================================
      SUBROUTINE DELAYD(DOTSK,OUT,PRIOR,ONOFF,MIN,SEC)
C=======================================================================
C this routine is called to turn a load off or on, after a specified time
C interval. DOTSK is the on-off task to use to control the load, OUT is
C the digital output to control, PRIOR is the priority to control the load
C under, ONOFF is 1 to turn on a load, 0 to turn it off, and MIN and SEC
C are the time interval that should elapse before the output is controlled.
C COMMON BLOCK ONOFF contains a table of on-off task parameters that are
C set before the task can be used.
```

```
C
      INTEGER PRIOR,OUT,DOTSK
      BYTE ONOFF,H,M,S,TS,TSK
      BYTE DONOFF,INUSE
      INTEGER MIN,SEC,IPR,DNUM
      BYTE CNTRL,IDOFF,SCAN0,SCMAST,SCSLAV,DCSLAV,TSKMAX,DCMAST
      COMMON/TSKINT/H,M,S,TS
      COMMON/ONOFF/DNUM(8),DONOFF(8),INUSE(8),IPR(8)
      COMMON/TSKLNK/CNTRL(16),IDOFF(16),SCAN0,SCMAST,SCSLAV(16),
     &DCSLAV(8),TSKMAX,DCMAST
C----------set parameters in on-off task table-------------------------
      INUSE(DOTSK)=1
      IPR(DOTSK)=PRIOR
      DNUM(DOTSK)=OUT
      DONOFF(DOTSK)=ONOFF
C----------cause on-off task to execute after a delay time-------------
      TS=0
      H=0
      M=MIN
      S=SEC
      TSK=DCSLAV(DOTSK)
      CALL TSKEDT(DCSLAV(DOTSK),-1,1)
      RETURN
      END
C::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
      SUBROUTINE ONOFF1
C::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
C This routine is one of several identical tasks used to turn off or
C turn on digital outputs after a short delay time. The output to be
C controlled by the task is programmable. There are multiple copies
C of these tasks to allow several outputs to be set up to turn off
C or on within the same future time frame.
C
      BYTE DONOFF,INUSE
      INTEGER IPR,DNUM
      BYTE CNTRL,IDOFF,SCAN0,SCMAST,SCSLAV,DCSLAV,TSKMAX,DCMAST
      COMMON/ONOFF/DNUM(8),DONOFF(8),INUSE(8),IPR(8)
C----------control digital output with priority IPR--------------------
      CALL DIGOUT(DNUM(1),DONOFF(1),IPR(1))
      RETURN
      END
```

## APPENDIX C - Duty Cycling Phase Allotment Algorithm

This appendix contains a detailed explanation, flow charts, and FORTRAN IV source code for a duty cycling table phase allotment algorithm (see section 4.4 of the main text). The routine might be used in central computer software (CCU) to assist the building operator in setting up duty cycling of building equipment.

The main flow diagram for the algorithm was given in figure 11. There are five passes that the algorithm makes over a duty cycle table containing a set of loads, and corresponding desired off-periods. The algorithm essentially fills in the phase parameters for each load to avoid simultaneous starting of equipment and to satisfy constraints on the loads of interlocking and exclusivity.

The first pass of the algorithm finds the load with the smallest off-period and the load with the longest off-period. The load with the smallest off-period is then assigned a phase of zero. The load with the largest off-period is given a phase which will cause the load to turn on 90% of the way through the duty cycle interval. If the remaining loads have small off-periods, their turn-on times can then be evenly spaced between the turn-on time of the load assigned the zero phase, and the turn on time of the load with the largest off-time, set at 90%. This is done by subtracting the off-period of the zero phase load (in %) from 90% to yield a range. The range is divided by the number of loads minus two, giving an increment for the on-times.

The second pass of the phase allotment algorithm tries to match load on-times with the ideal on-times determined after the first pass. A flag variable is used to indicate which loads have not been assigned a phase. The loads with the shortest and the longest off-periods were flagged as done after the first pass. The second pass tries to assign phases to all the remaining loads. A load on-time target is arrived at by subtracting the increment determined after the first pass from the previous target. The first value for the previous target is 90%. Thus if the increment was determined to be 10%, the targets would be 80%, 70%, 60%, etc. For each target, the algorithm loops through the table, looking for the load with the longest off-period that has not been assigned a phase. When this load is found, it is assigned a phase equal to the on-time target minus the off-period for the load. If the off-period is small, this value will be positive. It is possible for the calculated phase to be negative. This is allowable for the second pass because the negative phase will be corrected in the third pass. The algorithm continues assignment of values to phases to match targets until all of the loads have been assigned a phase. A flag is set if any of the phases were negative. If this flag is not set then the basic table is complete.

The third, fourth, and fifth passes of the phase allotment algorithm shown in figure 11 modify the basic duty cycling table for complex duty cycling

constraints. The third pass ensures that all phases that have been assigned are positive values. The fourth pass adjusts the table for interlocking between loads, and the fifth pass is used to satisfy mutually exclusive load constraints. If any of these types of constraints are absent, the third, fourth, or fifth passes may be skipped.

Figure A-1 is a flow chart illustrating the third pass of the phase allotment algorithm. This pass checks each phase, and if the phase is positive, makes no change. If a phase is negative (indicating a long off-period or similar problem) the algorithm uses a routine to determine what the nearness (in percent) of the ontime of the current load to other on-times in the table would be if the phase was assigned a value of zero. Also determined is the number of the load with the next chronological on-time. At this point, three constants must have been defined. These are a minimum spacing, which is the smallest desired time interval between two load on-times; a minimum separation, which is the smallest time interval between two on-times where another ontime may be inserted between these two; and the increment, which was determined during the second pass as the distance between on-time targets.

If the nearness factor is determined to be greater than the minimum separation, then the phase is made zero, and the next load in the table is checked. If the nearness is less than the minimum, another phase must be tried. The value of the next trial phase for the current load depends on how close the current load on-time is to the next load on-time in the table. If the space between these on-times is greater than the minimum space, the trial phase is increased by half the distance of the space between the current and next on-times. If the space between on-times is less than the minimum, the trial is increased by half the increment determined in the second pass. Using the new trial value, the nearness for the current load is checked again, and the phase is adjusted until all the tests described are passed.

If the trial phase reaches a point where the on-time value exceeds 95 percent, the trial is reset to zero and the minimum space and minimum separation constraints are halved. The process then continues until a valid positive phase can be assigned.

The fourth pass of the phase allotment algorithm consists of checks of the duty cycling table for entries in the column for interlocks between loads. If the load being checked is interlocked to another load, the table entry for the current load will contain a positive number which is the load number of the interlocked load. Figure A-2 is a chart illustrating the operation of the fourth pass of the algorithm. If the interlock load entry is indeed positive, a trial phase for the current load is set equal to the phase of the interlocked load. This will result in the two loads being off at the same time. The on-time of the current load must then be checked and its nearness to other on-times is determined with the same routine used by the third pass to check nearness. If the nearness is greater than the minimum separation, then the phase for the current load is made equal to the trial phase and the next load is checked for a positive load interlock. If the nearness is less

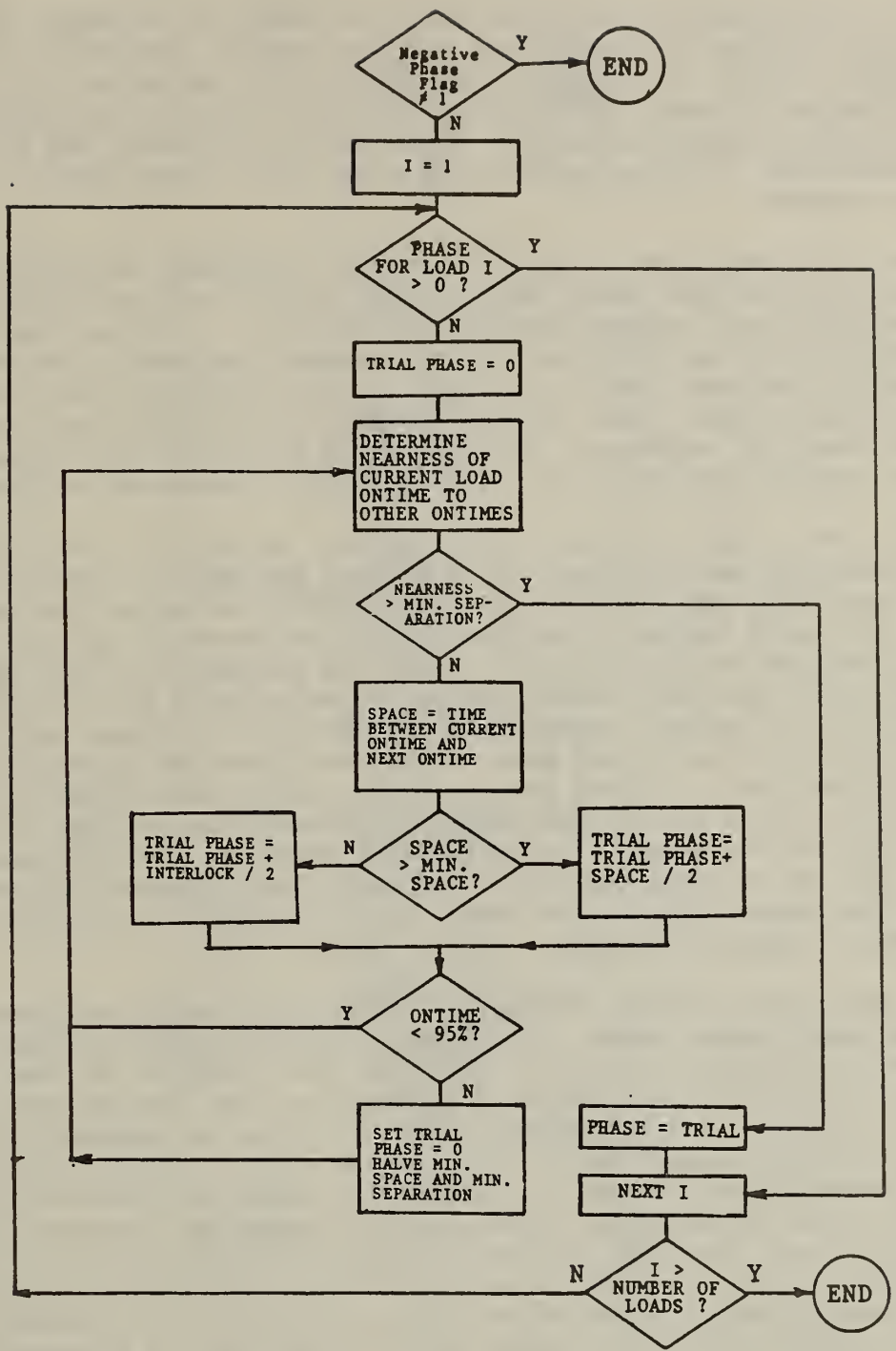


figure A-1. flow diagrams for the third pass of the phase allotment algorithm

than the minimum separation, the trial phase is increased by an amount equal to the minimum separation and the nearness is redetermined. Usually this is enough, unless the change in the trial phase has moved the on-time too close to another on-time. In this case the trial must be increased again. When all loads have been checked, the fourth pass is complete and all loads should be turned off at approximately the same time as any loads to which they are interlocked.

The fifth and final pass of the phase allotment algorithm checks the same column in the duty cycle table as the fourth pass, but only checks for values less than zero, rather than greater than zero. A value less than zero in the interlock column indicates that the current load should not be off at the same time as the load number specified by the absolute value of the number in the interlock column. Each load is checked to see if it has a negative number in the interlock column and if it does, the fifth pass must make an adjustment to the phase of the load. The fifth pass is illustrated by figure A-3.

When two loads are to be exclusive, there are two possible cases in determining the adjustment to be made to the phase of the current load to move it away from the interlocked load. The interlocked load may have its off-period start before the off-period of the current load, or the current load may have its off-period start before the off-period of the interlock load. In the first case, the current load would be moved in time so that its off-period would occur later, in order to increase the distance between the two off-periods. In the second case the current load off-period would be moved to occur earlier. A space between loads can be determined for each of the two cases, defined as the time interval between the off-time of one load and the on-time of the other. If the space is negative, this indicates that the loads are off at the same time during at least part of their off-periods. If either of the spaces for the two cases is greater than the minimum space, then no change is made to the phase of the current load.

If the spaces between off-periods for the two exclusive loads are not greater than the minimum space, a trial phase for the current load must be determined. First a decision is made to determine the direction to move the current load which would result in the least movement of the current load off-period. Once this is done, a trial phase can be calculated to separate the off-periods of the exclusive loads by the minimum separation. Using this trial phase, the nearness of the current load on-time to other on-times is determined, as was done in the previous two passes. If the nearness is greater than the minimum separation, then the current load phase is assigned the value of the trial phase. If the nearness is too small, the trial phase is either increased or decreased, depending on the relative positions of the current and interlocked load off-periods, by an amount equal to the minimum separation. The nearness is checked for the new phase until a the value is greater than the minimum separation.

At this point in the fifth pass, either the exclusive criterion for the current load has been satisfied, or the off-period of the current load has

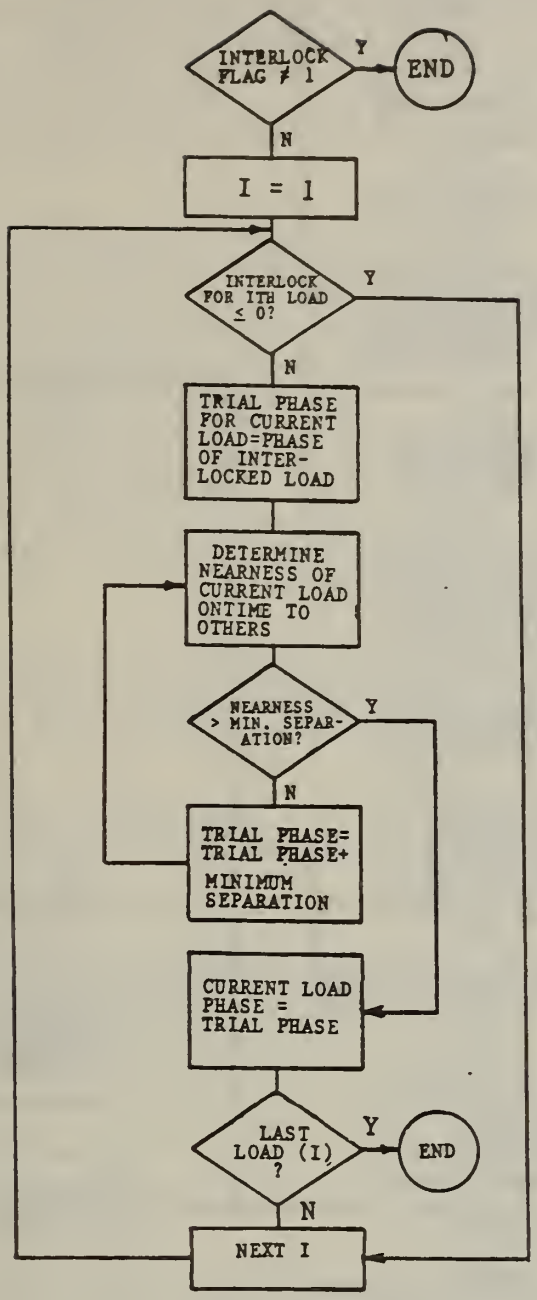


figure A-2. flow diagrams for the fourth pass of the phase allotment algorithm

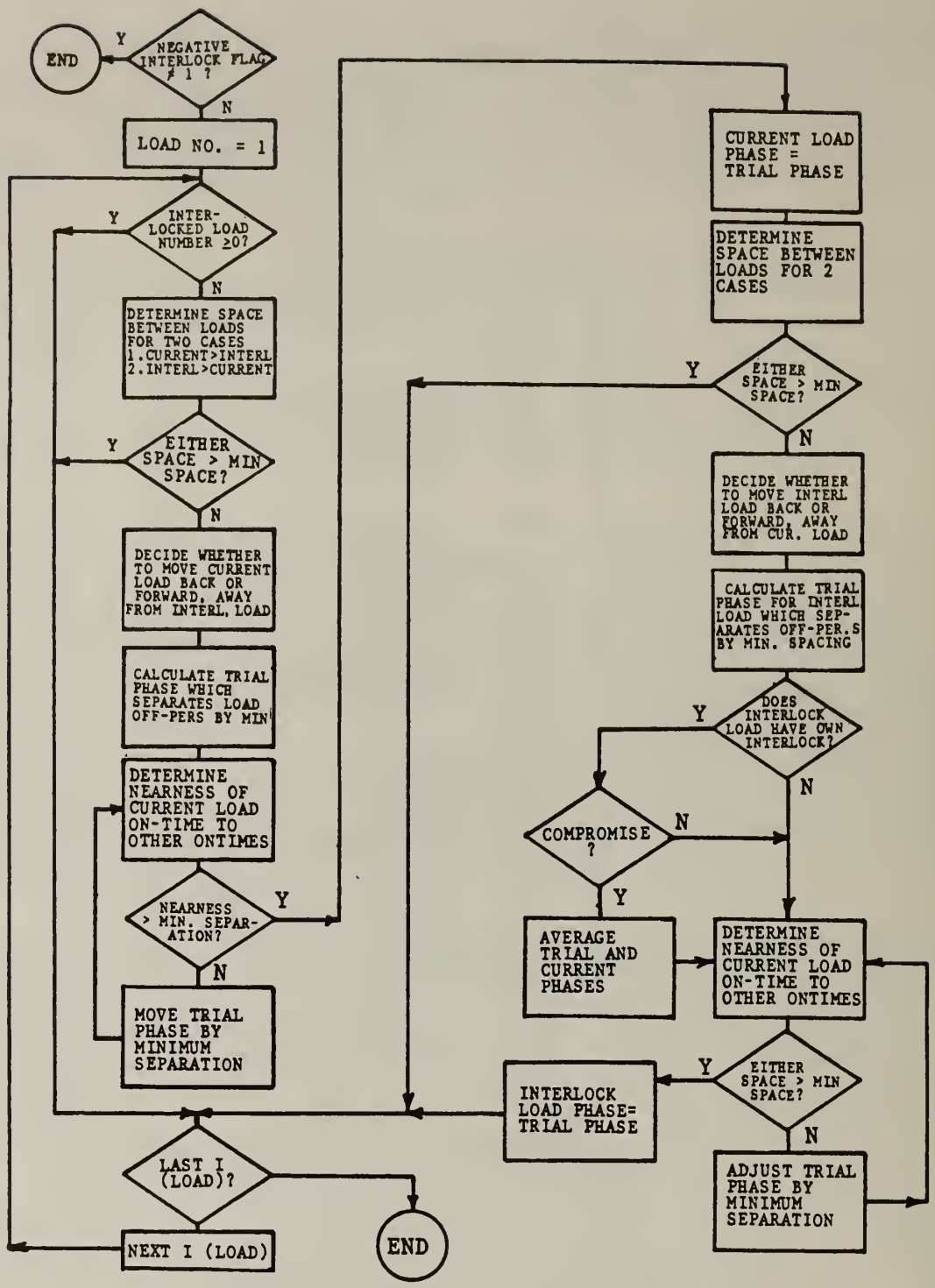


figure A-3. flow diagrams for the fifth pass of the phase allotment algorithm

been pushed all the way to an edge of the duty cycle interval. In the latter case, the off-periods of the current load and the interlock load may still not be exclusive. If this is true, then the phase of the interlock load can be adjusted. This adjustment is very similar to the phase adjustment for the current load, which has been discussed, except that the movement of the off-period is for the interlock load and spaces and separations are calculated relative to the current load.

Another difference between the exclusive loads phase adjustment for the interlock load and the current load is that the interlock load may be in turn interlocked to another load. Thus, when the trial phase for the interlock load is determined, it may be necessary to compromise the value for the trial phase. The decision to compromise depends on whether the movement of the interlock load off-period causes a conflict with the load off-period that the interlock load is itself interlocked with. If the interlock load off-period is moved away from an off-period that it is interlocked with, this is desirable and no compromise is necessary. Otherwise, a compromise is made. The compromise consists of halving the original change in phase that would have been made if there were no load interlocked to the interlock load of the current load.

Following is the listing of the duty cycling phase allotment algorithm, contained in the listing between label 1000 and label 4900. The other lines in the listing are a brief test program which the algorithm has been placed within.

```
C VERSION 0.28/SEPTEMBER 23,1982/W.B. MAY
C=================================================================
      PROGRAM DCPHASAL
C=================================================================
      INTEGER*1 DISPLA(40)
      INTEGER PCPHAS(25),PCTOFF(25),ONTIME,PDONE(25),NDEL,INTLOC(25)
      INTEGER NMIN,NMAX,MAXOFF,MINOFF,RANGE,TINCR,TARGET,DELTA,MINDEL
      LOGICAL NPFLAG,PIFLAG,NIFLAG
      INTEGER TRIAL,NLOW,NHIGH,NEARNS,SPACE,MINSEP,MINSPA,SPACE2,TRMAX
      DATA MINSEP/2/,MINSPA/5/
C-----------get off-time data-------------------------------
      WRITE (1,1)
    1 FORMAT(1X, 'DUTY CYCLING PHASE ALLOTMENT ALGORITHM TEST',//)
  100 DO 500 I=1,25
      WRITE(1,2)
    2 FORMAT(1H+,'ENTER PERCENT OFF TIME AND INTERLOCK FOR LOAD (-1=END)
     &',I3,'>')
      READ(1,3)PCTOFF(I),INTLOC(I)
C.................................................................
C INTERLOCK CODE: 0 = NO INTERLOCK
C                 >0 = INTERLOCK WITH LOAD OF SAME NUMBER
C                 <0 = MUTUALLY EXCLUSIVE WITH LOAD OF SAME NUMBER
C.................................................................
```

```
    3 FORMAT(2I10)
      IF(PCTOFF(I).LT.0.OR.PCTOFF(I).GT.99)GO TO 1000
      PCPHAS(I)=0
      PDONE(I)=0
  500 CONTINUE
C-----------------------------------------------------------------
C            phase allotment algorithm
C-----------------------------------------------------------------
 1000 N=I-1
      IF(N.EQ.1)GO TO 5000
C-----------first pass - find max and min offtimes--------------
      PIFLAG=0
      NIFLAG=0
      NPFLAG=0
      MINOFF=99
      MAXOFF=0
      DO 1200 I=1,N
      IF(INTLOC(I).LT.0)NIFLAG=1
      IF(INTLOC(I).GT.0)PIFLAG=1
      IF(PCTOFF(I).GT.MINOFF)GO TO 1100
      MINOFF=PCTOFF(I)
      NMIN=I
 1100 IF(PCTOFF(I).LT.MAXOFF)GO TO 1200
      MAXOFF=PCTOFF(I)
      NMAX=I
 1200 CONTINUE
      PCPHAS(NMAX)=90-MAXOFF
      RANGE=90-PCTOFF(NMIN)
      TINCR=RANGE/(N-1)
      PDONE(NMAX)=1
      PDONE(NMIN)=1
      IF(N.EQ.2)GO TO 3000
C-----------2ND pass - match loads to target turn on times---------
      N2=N-2
      TARGET=90
      DO 1500 I=1,N2
      MINDEL=0
      TARGET=TARGET-TINCR
      DO 1400 J=1,N
      IF(PDONE(J).EQ.1)GO TO 1400
      ONTIME=PCTOFF(J)+PCPHAS(J)
      IF(ONTIME.LT.MINDEL)GO TO 1400
      MINDEL=ONTIME
      NDEL=J
 1400 CONTINUE
      DELTA=TARGET-MINDEL
      PCPHAS(NDEL)=DELTA
      IF(DELTA.LT.0)NPFLAG=1
      PDONE(NDEL)=1
```

```
 1500 CONTINUE
C-------------3RD pass - negative phase removal---------------
 2000 IF(NPFLAG.NE.1)GO TO 3000
      DO 2500 I=1,N
      IF(PCPHAS(I).GE.0)GO TO 2500
      TRIAL=0
      MSEP=MINSEP
      MSPA=MINSPA
 2300 CALL ONTCHK(I,TRIAL,PCPHAS,PCTOFF,N,NLOW,NHIGH,NEARNS)
      IF(NEARNS.GE.MSEP)GO TO 2400
C----------nearness too small, adjust trial phase-------------
      SPACE=PCTOFF(NHIGH)+PCPHAS(NHIGH)-PCTOFF(I)-TRIAL
      IF(SPACE.GT.MSPA)TRIAL=TRIAL+SPACE/2
      IF(SPACE.LE.MSPA)TRIAL=TRIAL+TINCR/2
      ONTIME=TRIAL+PCTOFF(I)
      IF(ONTIME.LE.95)GO TO 2300
C---------exceed duty cycle interval, reduce minimums---------
      TRIAL=0
      MSEP=MSEP/2
      MSPA=MSPA/2
      GO TO 2300
 2400 PCPHAS(I)=TRIAL
 2500 CONTINUE
C-------------4TH pass - positive interlock adjustment---------
 3000 IF(PIFLAG.NE.1)GO TO 4000
      DO 3500 I=1,N
      IL=INTLOC(I)
      IF(IL.LE.0)GO TO 3500
      TRIAL=PCPHAS(IL)
 3300 CALL ONTCHK(I,TRIAL,PCPHAS,PCTOFF,N,NLOW,NHIGH,NEARNS)
      IF(NEARNS.GE.MINSEP)GO TO 3400
      TRIAL=TRIAL+MINSEP
      GO TO 3300
 3400 PCPHAS(I)=TRIAL
 3500 CONTINUE
C-------------5TH pass - exclusive loads adjustment------------
 4000 IF(NIFLAG.NE.1)GO TO 5000
      DO 4900 I=1,N
      IL=-INTLOC(I)
      IF(IL.LE.0)GO TO 4900
C------------check interload spacing --------------------------
      SPACE=PCPHAS(I)-PCTOFF(IL)-PCPHAS(IL)
      SPACE2=PCPHAS(IL)-PCTOFF(I)-PCPHAS(I)
      IF(SPACE.GE.MINSPA.OR.SPACE2.GE.MINSPA)GO TO 4900
      IF(SPACE2.GT.SPACE)GO TO 4200
C---------increase current load phase to clear interlock load---
      TRIAL=PCPHAS(IL)+PCTOFF(IL)+MINSPA
      TRMAX=95-PCTOFF(I)
      TRIAL=MIN0(TRIAL,TRMAX)
```

```
      MSEP=MINSEP
      GO TO 4300
C---------decrease current load phase to clear interlock load---
 4200 TRIAL=PCPHAS(IL)-PCTOFF(I)-MINSPA
      TRIAL=MAX0(TRIAL,0)
      MSEP=-MINSEP
 4300 CALL ONTCHK(I,TRIAL,PCPHAS,PCTOFF,N,NLOW,NHIGH,NEARNS)
      IF(NEARNS.GE.MINSEP)GO TO 4400
C-------slide off-time back or forward to clear conflict---------
      IF(TRIAL.LE.0.OR.TRIAL.GE.95)MSEP=-MSEP
      TRIAL=TRIAL-MSEP
      GO TO 4300
 4400 PCPHAS(I)=TRIAL
C---------if exclusive interlock not ok, move interlock load-------
      SPACE=PCPHAS(I)-PCTOFF(IL)-PCPHAS(IL)
      SPACE2=PCPHAS(IL)-PCTOFF(I)-PCPHAS(I)
      IF(SPACE.GE.MINSPA.OR.SPACE2.GE.MINSPA)GO TO 4900
      IF(SPACE2.GT.SPACE)GO TO 4420
C------decrease interlock load phase to clear current load--------
      TRIAL=PCPHAS(IL)-MINSPA-PCTOFF(I)
      TRIAL=MAX0(TRIAL,0)
      MSEP=-MINSEP
      GO TO 4450
C------increase interlock load phase to clear current load--------
 4420 TRIAL=PCPHAS(I)+PCTOFF(I)+MINSPA
      TRMAX=95-PCTOFF(IL)
      TRIAL=MIN0(TRIAL,TRMAX)
      MSEP=MINSEP
 4450 IF(INTLOC(IL).EQ.0)GO TO 4500
C---------if interlock load has interlock load, adjust trial------
      IF(INTLOC(IL).GT.0)GO TO 4470
      SPACE=TRIAL-PCPHAS(IL)
      IL2=INTLOC(IL)
      SPACE2=PCPHAS(IL)-PCPHAS(IL2)
      IF(SPACE.GT.0.AND.SPACE2.GT.0)GO TO 4500
      IF(SPACE.LT.0.AND.SPACE2.LT.0)GO TO 4500
 4470 TRIAL=(TRIAL+PCPHAS(IL))/2
 4500 CALL ONTCHK(IL,TRIAL,PCPHAS,PCTOFF,N,NLOW,NHIGH,NEARNS)
      TRMAX=95-PCTOFF(IL)
      IF(NEARNS.GE.MINSEP.OR.TRIAL.LE.0.OR.TRIAL.GE.TRMAX)GO TO 4600
      TRIAL=TRIAL+MSEP
      GO TO 4500
 4600 PCPHAS(IL)=TRIAL
 4900 CONTINUE
C-----------print out allotment-----------------------
 5000 WRITE(1,13)
   13 FORMAT(1X,//,1X,'   LOAD   PHASE  OFFTIME  TURN-ON',/)
      DO 5500 J=1,N
      ONTIME=PCTOFF(J)+PCPHAS(J)
```

```
      DO 5200 I=1,40
      DISPLA(I)=32
 5200 CONTINUE
      DISPLA(1)=124
      DISPLA(40)=124
      I1=PCPHAS(J)*10/25+1
      I2=ONTIME*10/25+1
      DO 5300 I=I1,I2
      DISPLA(I)=95
 5300 CONTINUE
      WRITE(1,4)J,PCPHAS(J),PCTOFF(J),ONTIME,INTLOC(J),DISPLA
    4 FORMAT(1X,I6,4I8,1X,40A1)
 5500 CONTINUE
      WRITE(1,15)
   15 FORMAT(1X,//)
      GO TO 100
      END
C=====================================================================
      SUBROUTINE ONTCHK(I,TRIAL,PCPHAS,PCTOFF,N,NLOW,NHIGH,NEARNS)
C=====================================================================
C This subroutine is used by the duty cycling phase allotment
C algorithm. Given a load with a trial phase and fixed off-time,
C the routine compares the ontime for this load with all other
C on-times and computes the nearness of the closest on time in %.
C The routine also points to the closest turn-on times on either
C side of the given load.
      INTEGER TRIAL,PCPHAS(25),PCTOFF(25),NLOW,NHIGH,NEARNS
      INTEGER PDEL,NDEL,DEL
      NLOW=1
      NHIGH=N
      PDEL=100
      NDEL=-100
      DO 1000 J=1,N
      IF(I.EQ.J)GO TO 1000
      IF(PCPHAS(J).LT.0)GO TO 1000
      DEL=TRIAL+PCTOFF(I)-PCPHAS(J)-PCTOFF(J)
      IF(DEL.LT.0)GO TO 400
      IF(DEL.GT.PDEL)GO TO 1000
      NLOW=J
      PDEL=DEL
      GO TO 1000
  400 IF(DEL.LT.NDEL)GO TO 1000
      NHIGH=J
      NDEL=DEL
 1000 CONTINUE
      NDEL=-NDEL
      NEARNS=MIN0(NDEL,PDEL)
      RETURN
      END
```

NBS-114A (REV. 2-8C)

| U.S. DEPT. OF COMM. BIBLIOGRAPHIC DATA SHEET (See Instructions) | 1. PUBLICATION OR REPORT NO. NBSIR 83-2713 | 2. Performing Organ. Report No. | 3. Publication Date July 1983 |
|---|---|---|---|

4. TITLE AND SUBTITLE

TIME OF DAY CONTROL AND DUTY CYCLING ALGORITHMS FOR BUILDING MANAGEMENT AND CONTROL SYSTEMS

5. AUTHOR(S)

William B. May, Jr.

6. PERFORMING ORGANIZATION (If joint or other than NBS, see instructions)

NATIONAL BUREAU OF STANDARDS
DEPARTMENT OF COMMERCE
WASHINGTON, D.C. 20234

7. Contract/Grant No.

8. Type of Report & Period Covered

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (Street, City, State, ZIP)

U.S. Department of Energy
1000 Independence Avenue, SW
Washington, DC 20585

Naval Civil Engineering Laboratory
Port Hueneme, CA 93043

10. SUPPLEMENTARY NOTES

☐ Document describes a computer program; SF-185, FIPS Software Summary, is attached.

11. ABSTRACT (A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here)

Software is an important component of building management and control systems (BMCS). Although much software is available in proprietary or system dependent form, public domain control software and algorithms are rare. This report describes concepts, algorithms, and software used in BMCS components developed in the NBS building systems and controls laboratory. The concepts and basic algorithms for time of day (scheduled start/stop) control and duty cycling of electrical equipment in building heating, ventilating, and air conditioning systems are presented. Time of day control results in control events occurring at predetermined times of the day on selected days of the week. Duty cycling is the periodic turning off and on of loads, usually electrical, to reduce energy consumption under part heating and cooling load conditions. Considerations for use of duty cycling with other control strategies such as demand limiting, selection of duty cycling parameters, and dynamic adjustment of duty cycling, are discussed. All algorithms presented were implemented in software on a specific BMCS, and the actual computer programs used are presented as examples.

12. KEY WORDS (Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons)

Building Management and Control Systems (EMCS, BMCS); computer control; control algorithms; control software; duty cycling; energy management; heating, ventilating and air conditioning (HVAC); scheduled start/stop; time of day control.

13. AVAILABILITY

☒ Unlimited
☐ For Official Distribution. Do Not Release to NTIS
☐ Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.
☒ Order From National Technical Information Service (NTIS), Springfield, VA. 22161

14. NO. OF PRINTED PAGES 60

15. Price $10.00

USCOMM-DC 6043-P80